TUGAS AKHIR – KI141502

DESAIN DAN ANALISIS ALGORITMA PENCARIAN PREDIKSI HASIL PENJUMLAHAN BEBERAPA URUTAN BERKALA DENGAN METODE ELIMINASI GAUSS

DANIEL HENRY
NRP. 5112 100 139

Dosen Pembimbing 1
Victor Hariadi, S.Si., M.Kom.

Dosen Pembimbing 2
Rully Soelaiman, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

TUGAS AKHIR – KI141502

DESAIN DAN ANALISIS ALGORITMA PENCARIAN PREDIKSI HASIL PENJUMLAHAN BEBERAPA URUTAN BERKALA DENGAN METODE ELIMINASI GAUSS

DANIEL HENRY
NRP. 5112 100 139

Dosen Pembimbing 1
Victor Hariadi, S.Si., M.Kom.

Dosen Pembimbing 2
Rully Soelaiman, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

(Halaman ini sengaja dikosongkan)

FINAL PROJECT – KI141502

DESIGN AND ANALYSIS OF ALGORITHM TO PREDICT SUM OF SEVERAL PERIODIC SEQUENCES USING GAUSS ELIMINATION

DANIEL HENRY
NRP. 5112 100 139

Supervisor 1
Victor Hariadi, S.Si., M.Kom.

Supervisor 2
Rully Soelaiman, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Sepuluh Nopember Institute of Technology
Surabaya 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

DESAIN DAN ANALISIS ALGORITMA PENCARIAN PREDIKSI HASIL PENJUMLAHAN BEBERAPA URUTAN BERKALA DENGAN METODE ELIMINASI GAUSS

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Dasar dan Terapan Komputasi
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
DANIEL HENRY
NRP. 5112100139

Disetujui oleh Pembimbing Tugas Akhir:

1. Victor Hariadi, S.Si., M.Kom. .....
   NIP. 196912281994121001 (Pembimbing 1)

2. Rully Soelaiman, S.Kom., M.Kom. .....
   NIP. 197002131994021001 (Pembimbing 2)

SURABAYA
JULI, 2017

(Halaman ini sengaja dikosongkan)

DESAIN DAN ANALISIS ALGORITMA PENCARIAN PREDIKSI HASIL PENJUMLAHAN BEBERAPA URUTAN BERKALA DENGAN METODE ELIMINASI GAUSS

Nama : **Daniel Henry**
NRP : **5112100139**
Jurusan : **Teknik Informatika**
Fakultas Teknologi Informasi ITS
Dosen Pembimbing I : **Victor Hariadi, S.Si., M.Kom.**
Dosen Pembimbing II : **Rully Soelaiman, S.Kom., M.Kom.**

ABSTRAK

Permasalahan dalam buku tugas akhir ini adalah permasalahan prediksi hasil penjumlahan beberapa urutan berkala. Dalam permasalahan ini, diberikan banyak urutan berkala N dimana panjang dari masing-masing urutan berkala berbeda satu dengan yang lainnya. Panjang dari urutan berkala dimulai dari N, N-1, N-2, hingga 1. Diberikan nilai f(0), f(1), f(2), hingga f(N^2-1), dimana f(x) didefinisikan sebagai penjumlahan tiap elemen N buah urutan berkala. Selanjutnya ditanyakan nilai f(x) dari nilai x yang diberikan.

Tugas akhir ini akan mengimplementasikan metode pencarian solusi sistem persamaan linear, yaitu metode eliminasi gauss.

Implementasi dalam tugas akhir ini menggunakan bahasa pemrograman C++. Hasil uji coba menunjukkan bahwa metode gauss eliminasi dapat menghasilkan jawaban permasalahan dengan benar, tetapi membutuhkan waktu yang sangat lama. Perlu adanya optimasi dengan mengubah permasalahan ke dalam bentuk interpolasi trigonometri yang diselesaikan dengan metode interpolasi polinomial Lagrange dan perkalian polinomial yang diselesaikan dengan metode transformasi Fourier cepat.

Kata kunci: Eliminasi Gauss, Sistem Persamaan Linear, Urutan Berkala.

DESIGN AND ANALYSIS OF ALGORITHM TO PREDICT SUM OF SEVERAL PERIODIC SEQUENCES USING GAUSS ELIMINATION

Name	**: Daniel Henry**
NRP	**: 5112100139**
Department	**: Department of Informatics**
	Faculty of Information Technology ITS
Supervisor I	**: Victor Hariadi, S.Si., M.Kom.**
Supervisor II	**: Rully Soelaiman, S.Kom., M.Kom.**

ABSTRACT

This final project is based on prediction of sum of several periodic sequences. In this problem, given N periodic sequences whose lengths are different each other. Lengths of periodic sequences start from N, N-1, N-2, until 1. Given values f(0), f(1), f(2), until $f(N^2-1)$*. f(x) is defined as the sum of N periodic sequences. Determine f(x) for all given x.*

This final project implements gauss elimination, which is an algorithm for solving linear equations system.

The implementation of final project uses C++ programming language. The experiment result proved gauss elimination could give correct prediction of sum of several periodic sequences, but took more time to get the answer. The solution must be optimized with converting the problem to trigonometric interpolation problem which be solved by Lagrange polynomial interpolation and polynomial multiplication which be solved by Fast Fourier Transform..

Keywords: Gauss Elimination, Linear Equations System, Periodic Sequence.

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan tugas akhir yang berjudul:

"DESAIN DAN ANALISIS ALGORITMA PENCARIAN PREDIKSI HASIL PENJUMLAHAN BEBERAPA URUTAN BERKALA DENGAN METODE ELIMINASI GAUSS"

Tugas akhir ini dilakukan untuk memenuhi salah satu syarat memperoleh gelar Sarjana Komputer di Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama proses pengerjaan tugas akhir ini hingga selesai, antara lain:

1. Tuhan Yang Maha Esa atas segala karunia dan rahmat-Nya yang telah diberikan selama ini.
2. Orang tua, saudara serta keluarga penulis yang selalu memberikan dukungan, semangat, perhatian dan doa selama perkuliahan penulis di Jurusan Teknik Informatika ini.
3. Bapak Rully Soelaiman, S.Kom., M.Kom. selaku dosen pembimbing II yang telah banyak memberikan ilmu, bimbingan, nasihat, motivasi, serta waktu diskusi sehingga penulis dapat menyelesaikan tugas akhir ini.
4. Bapak Victor Hariadi, S.Si., M.Kom. selaku dosen pembimbing I yang telah memberikan bimbingan, dukungan, motivasi, serta arahan dalam pengerjaan tugas akhir ini.

5. Kepada teman-teman penulis S1 Teknik Informatika ITS angkatan 2012 yang telah membantu, menyemangati, serta perhatian kepada penulis selama pengerjaan tugas akhir ini, terutama kepada Reva, Karsten, Anton, Ryan, Wik, Adhi, Uzi, Marvin, Prad, Vijay, Atank, Reyhan, Andrias, Deva, Tius, Peni, Otniel, Febrian, Christo, Andrys, Deni, Bian, Anggara, dan Wildi.
6. Kepada teman-teman penulis S1 Teknik Informatika ITS bukan angkatan 2012 yang juga telah banyak membantu, menyemangati, dan perhatian kepada penulis selama pengerjaan tugas akhir ini, terutama kepada Fendy, Ari, Dewangga, John, Juan, Daniel, Freddy, Andre, Nela, Gilang, Dave, Theo, Petrus, Steven, Bram, Lucas, Gerald, Glenn, Ajeng, Daus, dan Salma.
7. Kepada teman-teman penulis lainnya, yaitu Arie, Yehezkiel FT, Girang, Edwin, Hansel, dan Bamboy.
8. Seluruh pihak yang tidak bisa saya sebutkan satu persatu yang telah memberikan dukungan selama saya menyelesaikan tugas akhir ini.

Saya mohon maaf apabila terdapat kekurangan dalam penulisan buku tugas akhir ini. Kritik dan saran saya harapkan untuk perbaikan dan pembelajaran di kemudian hari. Semoga tugas akhir ini dapat memberikan manfaat yang sebaik-baiknya.

Surabaya, Juni 2017

Daniel Henry

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

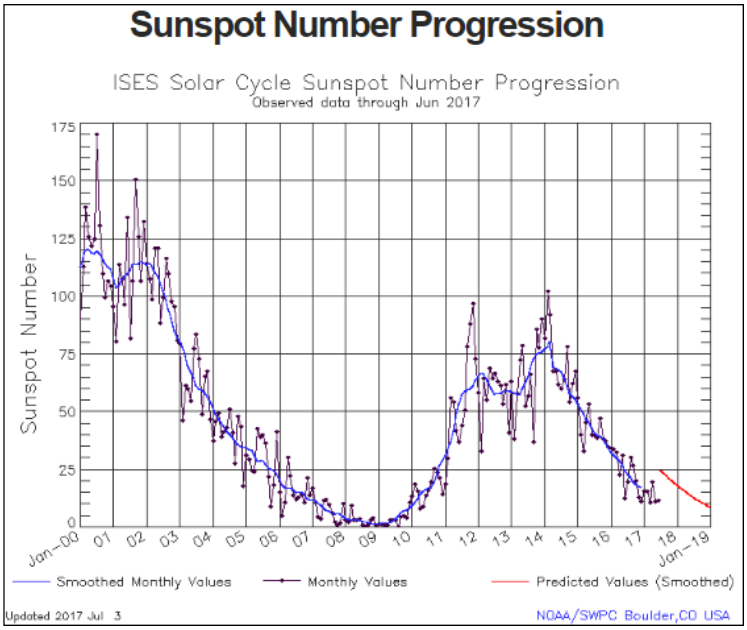
BAB 1
PENDAHULUAN

Pada bab ini akan dijelaskan hal-hal yang menjadi latar belakang, permasalahan yang dihadapi, batasan masalah, tujuan, metodologi dan sistematika penulisan yang digunakan dalam pembuatan buku tugas akhir ini.

1.1 Latar Belakang

Memprediksi suatu peristiwa merupakan suatu hal yang sangat penting, terutama peristiwa yang membutuhkan banyak biaya, baik dalam biaya tenaga maupun dalam biaya uang. Sebagai contoh, banyak perusahaan yang melakukan prediksi siklus matahari, khususnya memprediksi jumlah bintik matahari per bulan, untuk menentukan umur dari satelit buatan yang berada pada orbit bumi rendah dikarenakan umur satelit buatan bergantung pada siklus matahari [1]. Salah satu badan riset yang memprediksi jumlah bintik matahari per bulan yaitu *National Oceanic and Atmospheric Administration* (NOAA) milik Amerika Serikat. Data jumlah bintik matahari per bulan yang dimiliki oleh NOAA digunakan untuk memprediksi jumlah bintik matahari pada bulan berikutnya. Data jumlah bintik matahari pada bulan awal penelitian hingga bulan Juni tahun 2017 digunakan NOAA untuk memprediksi jumlah bintik matahari pada bulan Juli tahun 2017 hingga seterusnya yang terlihat pada Gambar 1.1 [2]. Dari hal ini, penulis tertarik untuk melihat proses prediksi siklus matahari.

Untuk memprediksi suatu nilai pada masa depan, dibutuhkan data nilai dari masa lalu. Penulis menggunakan deskripsi permasalahan, format data, serta batasan data yang ada pada situs *Sphere Online Judge* (SPOJ) dengan kode soal PERIOD4 [1]. Diberikan sebanyak N^2 data nilai bilangan bulat sebagai data masa lalu untuk memprediksi nilai masa depan. Data nilai merupakan hasil



Gambar 1.1 Jumlah Bintik Matahari per Bulan hingga Juni 2017

penjumlahan dari beberapa urutan berkala. Penjelasan mengenai urutan berkala dan kriteria urutan berkala yang digunakan akan dijelaskan lebih lanjut pada subbab 2.2.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana pemilihan dan desain algoritma untuk memprediksi hasil penjumlahan beberapa urutan berkala dengan memanfaatkan N^2 data bilangan bulat yang diketahui?
2. Bagaimana implementasi algoritma yang sudah didesain untuk memprediksi hasil penjumlahan beberapa urutan

berkala dengan memanfaatkan N^2 data bilangan bulat yang diketahui?

3. Bagaimana uji coba kebenaran dan uji coba kinerja prediksi penjumlahan beberapa urutan berkala?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Perangkat lunak yang dibuat hanya sebatas aplikasi konsol.
2. Masukan (*input*) dan keluaran (*output*) masing-masing menggunakan berkas *input* dan berkas *output*.
3. Implementasi dilakukan dengan bahasa pemrograman C++.
4. Banyak sub-kasus uji yang ada pada satu buah berkas *input* kasus uji antara 1 hingga 1024.
5. Nilai *N* berkisar antara 1 hingga 14.
6. Data nilai yang diketahui maupun yang akan diprediksi, berkisar antara -10^9 hingga 10^9.
7. Pengujian kebenaran hasil dilakukan dengan membandingkan 20 berkas *output* dari perangkat lunak dengan 20 berkas *output* dari *data generator*.

1.4 Tujuan

Tujuan dari pengerjaan tugas akhir ini adalah sebagai berikut:

1. Melakukan pemilihan dan desain algoritma untuk memprediksi hasil penjumlahan beberapa urutan berkala dengan memanfaatkan N^2 data bilangan bulat yang diketahui.

2. Melakukan implementasi algoritma yang sudah dipilih dan didesain untuk melakukan prediksi hasil penjumlahan beberapa urutan berkala.
3. Melakukan uji coba untuk mengetahui kebenaran hasil keluaran (*output*) dan uji coba kinerja dari implementasi yang telah dilakukan.

1.5 Metodologi

Ada beberapa tahapan dalam pengerjaan tugas akhir ini, yaitu sebagai berikut:

1. Penyusunan Proposal Tugas Akhir
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Pada proposal ini, penulis mengajukan gagasan untuk menyelesaikan permasalahan prediksi hasil penjumlahan beberapa urutan berkala.
2. Studi Literatur
   Pada tahap ini dilakukan studi literatur yang membahas lebih dalam yang berkaitan dengan eliminasi gauss. Studi literatur didapatkan dari buku, internet, dan materi-materi kuliah yang berhubungan dengan metode yang akan digunakan.
3. Desain
   Pada tahap ini akan dibahas mengenai desain algoritma berdasarkan studi literatur yang telah diuraikan sebelumnya.
4. Implementasi
   Pada tahap ini dilakukan implementasi algoritma berdasarkan analisis dan desain sebelumnya. Implementasi akan dilakukan dengan bahasa pemrograman C++ dan menggunakan IDE Dev-C++.

5. Pengujian dan evaluasi
   Pada tahap ini dilakukan uji coba kebenaran hasil keluaran dari solusi yang telah diimplementasi dengan membandingkan 30 berkas *output* dari perangkat lunak solusi dengan 30 berkas *output* dari *data generator*. Uji coba kinerja dilakukan dengan melihat waktu berjalan (*running time*) implementasi perangkat lunak dengan berkas masukan yang dibuat oleh *data generator*.
6. Penyusunan buku tugas akhir
   Tahap ini merupakan tahap penyusunan laporan berupa buku sebagai dokumentasi pengerjaan tugas akhir yang mencakup seluruh dasar teori, desain, implementasi serta hasil pengujian yang telah dilakukan.

1.6 Sistematika Penulisan

Penulisan buku tugas akhir ini dibagi kedalam 6 bab yang masing-masing membahas bagian-bagian yang disusun dalam menyelesaikan persoalan yang terkait, yaitu:

1. Bab 1: Pendahuluan
   Bab ini berisi penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi, dan sistematika penulisan buku.
2. Bab 2: Dasar Teori
   Bab ini berisi penjelasan mengenai teori-teori yang digunakan sebagai dasar pengerjaan tugas akhir ini.
3. Bab 3: Desain
   Bab ini berisi rancangan pembuatan perangkat lunak penyelesaian permasalahan dalam tugas akhir ini.
4. Bab 4: Implementasi
   Bab ini berisi lingkungan serta hasil penerapan rancangan perangkat lunak penyelesaian permasalahan

dalam tugas akhir ini dalam bentuk kode sumber beserta penjelasannya.

5. Bab 5: Uji Coba dan Evaluasi
   Bab ini berisi lingkungan serta hasil dari rangkaian uji coba yang dilakukan untuk menguji kebenaran hasil keluaran.
6. Bab 6: Kesimpulan dan Saran
   Bab ini berisi kesimpulan pengerjaan tugas akhir dan saran untuk pengembangan kedepannya.

BAB 2
DASAR TEORI

Bab ini akan membahas tentang definisi umum, deskripsi permasalahan, contoh kasus permasalahan, literatur, serta cara penerapan literatur pada tugas akhir ini.

2.1 Definisi Umum

Dalam subbab ini membahas definisi-definisi umum yang akan digunakan sebagai dasar untuk memahami soal dan menyelesaikan permasalahan ini.

2.1.1 Urutan Berkala

Urutan berkala (*periodic sequence*) [3] adalah suatu urutan $s = (s_0, s_1, s_2, \dots)$ dimana $s_i = s_{i+N}$ untuk semua $i \geq 0$. Urutan s juga bisa disebut sebagai urutan N-berkala, dimana N merupakan nilai periode dari urutan s. Sebagai contoh, urutan $U = (2, 3, 4, 2, 3, 4, 2, 3, 4, \dots)$ merupakan sebuah urutan 3-berkala dan urutan $V = (5, 4, 2, 3, 5, 4, 2, 3, 5, 4, 2, 3, \dots)$ merupakan sebuah urutan 4-berkala.

2.1.2 Persamaan Linear

Persamaan linear [4] dalam variabel x dan y didefinisikan sebagai sebuah garis lurus pada bidang datar xy yang dapat direpresentasikan dengan persamaan (2, 1) dimana a_1, a_2, dan b merupakan konstanta real serta a_1 dan a_2 tidak bernilai 0.

$$a_1 x + a_2 y = b \qquad (2, 1)$$

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = b \qquad (2, 2)$$

Secara umum, persamaan linear dalam n variabel x_1, x_2, ..., x_n dapat ditulis ke dalam bentuk persamaan (2, 2) dimana a_1, a_2, ..., a_n, dan b merupakan konstanta real.

2.1.3 Sistem Persamaan Linear

Sistem persamaan linear [4] adalah kumpulan dari dua atau lebih persamaan linear dalam variabel x_1, x_2, ..., x_n. Sebuah urutan angka $s = (s_1, s_2, \dots, s_n)$ disebut sebagai solusi dari suatu sistem persamaan linear jika $x_1 = s_1, x_2 = s_2$, ..., $x_n = s_n$ adalah solusi dari setiap persamaan linear dalam sistem.

$$4x_1 - x_2 + 3x_3 = -1 \qquad (2, 3)$$

$$3x_1 + x_2 + 9x_3 = -4 \qquad (2, 4)$$

Sebagai contoh, persamaan (2, 3) dan (2, 4) merupakan dua persamaan linear yang berada dalam suatu sistem persamaan linear, dimana memiliki solusi $x_1 = 1, x_2 = 2, x_3 = -1$. Solusi ini dapat menyelesaikan persamaan (2, 3) dan persamaan (2, 4). Namun, $x_1 = 1, x_2 = 8$, $x_3 = 1$ bukan merupakan solusi sistem persamaan karena hanya menjadi solusi dari persamaan (2, 3).

2.2 Deskripsi Permasalahan

Permasalahan yang diangkat dalam tugas akhir ini mengangkat permasalahan prediksi hasil penjumlahan beberapa urutan berkala. Pada permasalahan ini, diberikan beberapa buah berkas kasus uji dimana dalam sebuah berkas terdapat beberapa sub-kasus uji T. Untuk setiap sub-kasus uji, diberikan sebuah bilangan N yang menandakan banyaknya urutan berkala yang terlibat dan N^2 bilangan bulat yang merepresentasikan data di masa lalu. N^2 bilangan tersebut merupakan nilai dari $f(0)$, $f(1)$, $f(2)$, hingga $f(N^2 - 1)$. Definisi $f(x)$ itu sendiri merupakan penjumlahan dari N buah urutan berkala. Untuk nilai $N = 2$, $f(x)$ merupakan

Tabel 2.1 Definisi f(x) untuk N=2 dengan 4 Nilai f(x) Diketahui

x	**Definisi f**	$f(x)$
0	$b_0 + c_0$	$f(0)$
1	$b_1 + c_0$	$f(1)$
2	$b_0 + c_0$	$f(2)$
3	$b_1 + c_0$	$f(3)$
4	$b_0 + c_0$	??
5	$b_1 + c_0$	??
6	$b_0 + c_0$	??
…	…	…

Tabel 2.2 Definisi f(x) untuk N=3 dengan 9 Nilai f(x) Diketahui

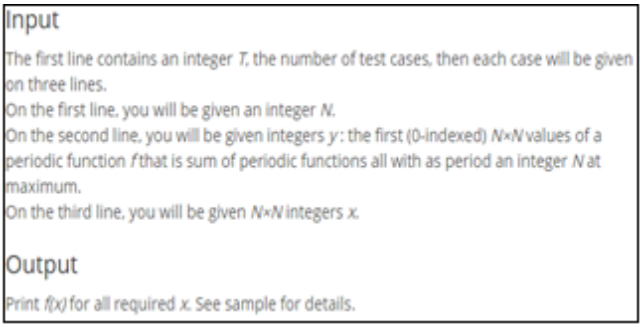
x	**Definisi f**	$f(x)$
0	$a_0 + b_0 + c_0$	$f(0)$
1	$a_1 + b_1 + c_0$	$f(1)$
2	$a_2 + b_0 + c_0$	$f(2)$
3	$a_0 + b_1 + c_0$	$f(3)$
4	$a_1 + b_0 + c_0$	$f(4)$
5	$a_2 + b_1 + c_0$	$f(5)$
6	$a_0 + b_0 + c_0$	$f(6)$
7	$a_1 + b_1 + c_0$	$f(7)$
8	$a_2 + b_0 + c_0$	$f(8)$
9	$a_0 + b_1 + c_0$	??
10	$a_1 + b_0 + c_0$	??
11	$a_2 + b_1 + c_0$	??
12	$a_0 + b_0 + c_0$	??
…	…	…

penjumlahan dari urutan 2-berkala dan urutan 1-berkala dengan 4 data nilai diketahui seperti yang tertera pada Tabel 2.1.

Untuk nilai $N = 3$, $f(x)$ merupakan penjumlahan dari urutan 3-berkala, urutan 2-berkala, dan urutan 1-berkala dengan 9 data nilai diketahui seperti yang tertera pada Tabel 2.2. Untuk nilai $N \geq 4$, berlaku hal yang sama seperti $N = 2$ dan $N = 3$, dimana $f(x)$ merupakan penjumlahan dari urutan N-berkala, urutan $(N-1)$-berkala, $(N-2)$-berkala, hingga urutan 1-berkala dengan N^2 data nilai diketahui. Selanjutnya, sebanyak N^2 ditanyakan nilai $f(x)$ dari nilai x yang diberikan.

Nilai-nilai dari $f(x)$ membentuk sebuah urutan berkala, dimana nilai berkalanya bergantung pada nilai N. Untuk nilai $N = 2$, nilai-nilai dari $f(x)$ membentuk urutan 2-berkala. Sedangkan, untuk nilai $N = 3$, nilai-nilai dari $f(x)$ membentuk urutan 6-berkala. Sehingga, dapat disimpulkan bahwa penjumlahan N urutan berkala akan membentuk sebuah urutan berkala, dimana nilai berkalanya merupakan kelipatan persekutuan terkecil (KPK) dari semua nilai berkala urutan berkala yang dijumlahkan. Untuk nilai $N = 2$, urutan 2-berkala didapat dari penjumlahan urutan 2-berkala dan urutan 1-berkala, dimana KPK dari 2 dan 1 adalah 2. Sedangkan, untuk nilai $N = 3$, urutan 6-berkala didapat dari penjumlahan urutan 3-berkala, urutan 2-berkala, dan urutan 1-berkala, dimana KPK dari 3, 2, dan 1 adalah 6. Tabel 2.3 menunjukkan nilai berkala dari urutan berkala yang dihasilkan oleh penjumlahan urutan N-berkala, urutan $(N-1)$-berkala, urutan $(N-2)$-berkala, hingga urutan 1-berkala.

Tabel 2.4 menunjukkan bahwa banyak nilai $f(x)$ yang diketahui tidak selalu melebihi atau sama dengan nilai berkala urutan berkala $f(x)$. Untuk $N < 5$, banyak nilai $f(x)$ yang diketahui lebih besar dari nilai berkala urutan berkala $f(x)$. Untuk $N \geq 5$, banyak nilai $f(x)$ yang diketahui lebih kecil dari nilai berkala urutan berkala $f(x)$. Misalkan P merupakan nilai berkala urutan berkala $f(x)$. Maka, perlu adanya prediksi nilai $f(N^2)$ hingga nilai $f(P-1)$.



Input

The first line contains an integer *T*, the number of test cases, then each case will be given on three lines.
On the first line, you will be given an integer *N*.
On the second line, you will be given integers *y*: the first (0-indexed) *N*×*N* values of a periodic function *f* that is sum of periodic functions all with as period an integer *N* at maximum.
On the third line, you will be given *N*×*N* integers *x*.

Output

Print *f(x)* for all required *x*. See sample for details.

Gambar 2.1 Format Masukan dan Keluaran pada Situs SPOJ

Tabel 2.3 Nilai Berkala dari Urutan Berkala f(x) tiap Nilai N

N	**Nilai Berkala Urutan Berkala Hasil Penjumlahan Urutan [*N*, … , 2, 1]-Berkala**
1	1
2	2
3	6
4	12
5	60
6	60
7	420
8	840
9	2520
10	2520
11	27720
12	27720
13	360360
14	360360

Format masukan dan keluaran yang digunakan mengikuti format masukan dan keluaran pada situs SPOJ dengan judul soal *Periodic function, trip 3 (easy)* [3] seperti yang tertera pada Gambar 2.1.

```
Input:
2
2
5 7 5 7
6 7 8 9
3
15 3 17 2 16 4 15 3 17
10 100 1000 10000 100000 1000000 10000000 100000000 1000000000

Output:
5 7 5 7
16 16 16 16 16 16 16 16 16
```

Gambar 2.2 Contoh Masukan dan Keluaran pada Situs SPOJ

```
0 < T < 1024
1 < N < 14 : uniform distribution
abs(y) < 10^9
0 < X < 10^9
```

Gambar 2.3 Batasan Tugas Akhir yang Diambil dari Situs SPOJ

Contoh masukan dan keluaran yang digunakan mengikuti contoh masukan dan keluaran pada situs SPOJ dengan judul soal yang sama seperti yang tertera pada Gambar 2.2.

Batasan untuk tugas akhir ini juga diambil dari situs yang sama dengan judul soal yang sama seperti yang tertera pada Gambar 2.3.

Batasan tugas akhir yang diambil dari situs SPOJ adalah sebagai berikut:

- $0 < T < 1024$
- $1 < N < 14$
- $1000000000 < y < 1000000000$
- $0 < x < 1000000000$

Tabel 2.4 Perbandingan Banyak Nilai f(x) yang Diketahui dengan Nilai Berkala Urutan Berkala f(x)

N	**Banyak Nilai $f(x)$ yang Diketahui**	**Nilai Berkala Urutan Berkala $f(x)$**
1	1	1
2	4	2
3	9	6
4	16	12
5	25	60
6	36	60
7	49	420
8	64	840
9	81	2520
10	100	2520
11	121	27720
12	144	27720
13	169	360360
14	196	360360

Nilai $f(x)$ yang ditanyakan berkisar antara $f(0)$ hingga $f(1000000000)$. Karena semua nilai $f(x)$ berulang sesuai dengan nilai berkalanya, maka nilai $f(x)$ yang ditanyakan dapat dimampatkan menjadi berkisar antara $f(0)$ hingga $f(P-1)$, dimana P adalah nilai berkala urutan berkala $f(x)$. Prediksi dilakukan bukan untuk mencari nilai $f(N^2)$ hingga $f(1000000000)$, melainkan mencari nilai $f(N^2)$ hingga $f(P-1)$, Karena nilai $f(P)$ hingga $f(1000000000)$ telah terdapat pada nilai $f(0)$ hingga $f(P-1)$. Untuk beberapa nilai N, prediksi akan sulit untuk dilakukan karena semakin tinggi nilai N, semakin besar selisih antara banyak nilai $f(x)$ yang diketahui dan nilai P.

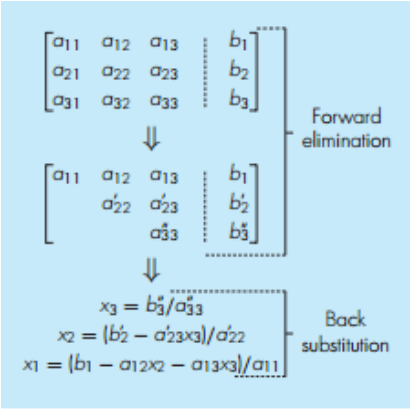
2.3 Contoh Kasus Permasalahan

Sebagai contoh, jika diberikan N dengan nilai 2, maka akan diketahui grafik dengan 4 nilai titik dengan nilai $f(0) = 5, f(1) = 7, f(2) = 5,$ dan $f(3) = 7,$ dimana definisi dari f tiap titik tertera pada Tabel 2.5. Terdapat 2 buah urutan berkala pada contoh ini, yaitu $b = (b_0, b_1, b_0, b_1, \dots),$ dan $c = (c_0, c_0, c_0, c_0, \dots),$ dimana b merupakan urutan 2-berkala dan c merupakan urutan 1-berkala yang belum diketahui nilainya dan perlu dicari nilainya untuk bisa mencari nilai f(x).

Tabel 2.5 Contoh Kasus untuk N=2

x	Definisi f	$f(x)$
0	$b_0 + c_0$	5
1	$b_1 + c_0$	7
2	$b_0 + c_0$	5
3	$b_1 + c_0$	7
4	$b_0 + c_0$	??
5	$b_1 + c_0$	??
6	$b_0 + c_0$	??
…	…	…

Contoh lainnya, jika diberikan N dengan nilai 3, maka akan diketahui grafik dengan 9 nilai titik dengan nilai $f(0) = 15, f(1) = 3, f(2) = 17, f(3) = 2, f(4) = 16, f(5) = 4, f(6) = 15, f(7) = 3,$ dan $f(8) = 17,$ dimana definisi dari f tiap titik tertera pada Tabel 2.6. Terdapat 3 buah urutan berkala pada contoh ini, yaitu $a = (a_0, a_1, a_2, a_0, a_1, a_2, \dots),$ $b = (b_0, b_1, b_0, b_1, b_0, b_1, \dots),$ dan $c = (c_0, c_0, c_0, c_0, c_0, c_0, \dots),$ dimana a merupakan urutan 3-berkala, b merupakan urutan 2-berkala, dan c merupakan urutan 1-berkala yang belum diketahui nilainya dan perlu dicari nilainya untuk bisa mencari nilai f(x).



Gambar 2.4 Ilustrasi Singkat Proses Eliminasi Gauss

Tabel 2.6 Contoh Kasus untuk N=3

x	**Definisi f**	$f(x)$
0	$a_0 + b_0 + c_0$	15
1	$a_1 + b_1 + c_0$	3
2	$a_2 + b_0 + c_0$	17
3	$a_0 + b_1 + c_0$	2
4	$a_1 + b_0 + c_0$	16
5	$a_2 + b_1 + c_0$	4
6	$a_0 + b_0 + c_0$	15
7	$a_1 + b_1 + c_0$	3
8	$a_2 + b_0 + c_0$	17
9	$a_0 + b_1 + c_0$	??
10	$a_1 + b_0 + c_0$	??
11	$a_2 + b_1 + c_0$	??
12	$a_0 + b_0 + c_0$	??
…	…	…

2.4 Eliminasi Gauss

Eliminasi gauss [5] adalah suatu algoritma yang digunakan untuk mencari solusi dari suatu sistem persamaan linear. Sistem persamaan linear (2, 5) diubah ke dalam bentuk perkalian matriks, sehingga berubah bentuk menjadi (2, 6).

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \qquad\qquad & \quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n &= b_n \end{aligned} \qquad (2, 5)$$

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}}_{C} \qquad (2, 6)$$

Dalam proses eliminasi gauss, matriks A dilakukan operasi perkalian, penjumlahan, pengurangan, dan/atau pembagian baris matriks hingga terbentuk matriks A' pada persamaan (2, 7).

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a'_{22} & a'_{23} & \dots & a'_{2n} \\ 0 & 0 & a'_{33} & \dots & a'_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a'_{nn} \end{bmatrix}}_{A'} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} b_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_n \end{bmatrix}}_{C'} \qquad (2, 7)$$

Selanjutnya, dilakukan proses *back substitution* agar mendapatkan solusi dari sistem persamaan linear. Gambar 2.4 menunjukkan proses eliminasi gauss dari awal sampai proses *back substitution*.

Kompleksitas algoritma eliminasi gauss adalah $O(n^3)$ dengan n adalah banyaknya variabel pada satu persamaan linear pada sistem persamaan linear. Kompleksitas dari proses *back substitution* yaitu $O(n^2)$ dengan n adalah banyaknya variabel. Total kompleksitas dari keseluruhan proses eliminasi gauss adalah $O(n^3)$.

2.5 Penyelesaian Permasalahan

Untuk nilai N = 2, definisi f yang tertera pada Tabel 2.5 dapat diubah ke dalam bentuk sistem persamaan linear pada Tabel 2.7 dengan memasukkan semua variabel pada semua urutan berkala yang tersedia. Dalan kasus N = 2, banyak urutan berkala adalah dua buah urutan, yaitu urutan 2-berkala berisi 2 variabel dan urutan 1-berkala berisi 1 variabel dengan total jumlah variabel adalah 3 variabel. Jumlah persamaan linear yang digunakan hanya 3 persamaan pertama dari 4 persamaan linear yang ada karena mengikuti total jumlah variabel yang digunakan.

Tabel 2.7 Persamaan Linear Awal untuk N=2

x	**Persamaan Linear $f(x)$**
0	$1b_0 + 0b_1 + 1c_0 = 5$
1	$0b_0 + 1b_1 + 1c_0 = 7$
2	$1b_0 + 0b_1 + 1c_0 = 5$

Untuk nilai N = 3, definisi f yang tertera pada Tabel 2.6 dapat diubah ke dalam bentuk sistem persamaan linear pada Tabel 2.8 dengan memasukkan semua variabel pada semua urutan berkala yang tersedia. Dalan kasus N = 3, banyak urutan berkala adalah tiga buah urutan, yaitu urutan 3-berkala berisi 3 variabel, urutan 2-berkala berisi 2 variabel, dan urutan 1-berkala berisi 1 variabel dengan total jumlah variabel adalah 6 variabel. Jumlah persamaan linear yang digunakan hanya 6 persamaan pertama dari 9 persamaan linear yang ada karena mengikuti total jumlah variabel yang digunakan.

Tabel 2.8 Persamaan Linear Awal untuk N=3

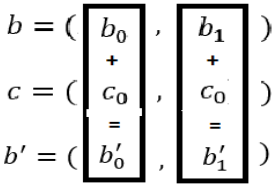
x	**Persamaan Linear $f(x)$**
0	$1c_0 + 0c_1 + 0c_2 + 1b_0 + 0b_1 + 1c_0 = 15$
1	$0c_0 + 1c_1 + 0c_2 + 0b_0 + 1b_1 + 1c_0 = 3$
2	$0c_0 + 0c_1 + 1c_2 + 1b_0 + 0b_1 + 1c_0 = 17$
3	$1c_0 + 0c_1 + 0c_2 + 0b_0 + 1b_1 + 1c_0 = 2$
4	$0c_0 + 1c_1 + 0c_2 + 1b_0 + 0b_1 + 1c_0 = 16$
5	$0c_0 + 0c_1 + 1c_2 + 0b_0 + 1b_1 + 1c_0 = 4$

Namun, jumlah variabel awal untuk nilai *N* pada batasan $1 \leq N \leq 14$ yang tertera pada Tabel 2.9 masih dapat dikurangi untuk sedikit mempercepat proses eliminasi gauss pada pencarian nilai-nilai variabel.

Banyak variabel dapat dikurangi dengan cara menjumlahkan variabel pada q-urutan berkala dengan variabel pada qr-urutan berkala sehingga menghasilkan qr-urutan berkala baru, dimana q merupakan bilangan bulat positif dan r merupakan bilangan bulat lebih besar dari 1.

Sebagai contoh, pada Gambar 2.5 variabel yang terdapat pada urutan 1-berkala c dapat dijumlahkan dengan variabel pada urutan 2-berkala b sehingga menghasilkan urutan 2-berkala baru b'. Dalam contoh ini, nilai q adalah 1 dan nilai r adalah 2.

Untuk N = 4, terdapat 4 buah urutan berkala, yaitu urutan 4-berkala, urutan 3-berkala, urutan 2-berkala, dan urutan 1-berkala. Urutan 1-berkala dapat ditambahkan ke dalam urutan 2-berkala dan membentuk urutan 2-berkala baru seperti pada Gambar 2.5. Selanjutnya, urutan 2-berkala baru dapat ditambahkan ke dalam urutan 4-berkala sehingga membentuk urutan 4-berkala baru seperti yang tertera pada Gambar 2.6. Banyak variabel akhir yang dibutuhkan jika N = 4 adalah sebanyak 7 variabel. Tabel 2.10

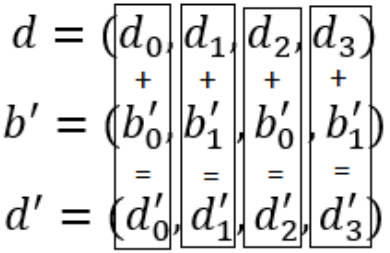


Gambar 2.5 Ilustrasi Penambahan Urutan 1-Berkala ke dalam Urutan 2-Berkala

Tabel 2.9 Banyak Variabel Awal Setiap Nilai N

Banyak Urutan Berkala Awal (*N*)		**Banyak Variabel Awal**
1	1	1
2	2+1	3
3	3+2+1	6
4	4+3+2+1	10
5	5+4+3+2+1	15
6	6+5+4+3+2+1	21
7	7+6+5+4+3+2+1	28
8	8+7+6+5+4+3+2+1	36
9	9+8+7+6+5+4+3+2+1	45
10	10+9+8+7+6+5+4+3+2+1	55
11	11+10+9+8+7+6+5+4+3+2+1	66
12	12+11+10+9+8+7+6+5+4+3+2+1	78
13	13+12+11+10+9+8+7+6+5+4+3+2+1	91
14	14+13+12+11+10+9+8+7+6+5+4+3+2+1	105

menunjukkan banyak variabel akhir yang dibutuhkan oleh setiap nilai N pada selang $1 \leq N \leq 14$.



Gambar 2.6 Ilustrasi Penambahan Urutan 2-Berkala ke dalam Urutan 4-Berkala

Tabel 2.10 Banyak Variabel Akhir Setiap Nilai N

Banyak Urutan Berkala Awal (*N*)		**Banyak Urutan Berkala Akhir (*K*)**	**Banyak Variabel Akhir (*W*)**
1	1	1	1
2	2	1	2
3	3+2	2	5
4	4+3	2	7
5	5+4+3	3	12
6	6+5+4	3	15
7	7+6+5+4	4	22
8	8+7+6+5	4	26
9	9+8+7+6+5	5	35
10	10+9+8+7+6	5	40
11	11+10+9+8+7+6	6	51
12	12+11+10+9+8+7	6	57
13	13+12+11+10+9+8+7	7	70
14	14+13+12+11+10+9+8	7	77

Banyak persamaan linear yang akan diselesaikan dengan eliminasi gauss mengikuti banyak variabel akhir seperti yang tertera pada Tabel 2.11 dimana banyak persamaan linear yang digunakan lebih sedikit dari total persamaan linear yang diketahui.

Tabel 2.11 Banyak Persamaan Linear Setiap Nilai N

N	Banyak Variabel (W) Akhir	Banyak Persamaan Linear yang Digunakan	Banyak Persamaan Linear yang Diketahui (N^2)
1	1	1	1
2	2	2	4
3	5	5	9
4	7	7	16
5	12	12	25
6	15	15	36
7	22	22	49
8	26	26	64
9	35	35	81
10	40	40	100
11	51	51	121
12	57	57	144
13	70	70	169
14	77	77	196

Untuk mencari banyak variabel akhir untuk semua nilai N seperti yang tertera pada Tabel 2.11, digunakan rumus penjumlahan deret aritmatika pada persamaan (2, 8), dimana W merupakan banyak variabel akhir, $P_{terpendek}$ merupakan panjang urutan berkala terpendek yang dicari dengan rumus (2, 10), $P_{terpanjang}$ merupakan panjang urutan berkala terpanjang yang dicari dengan rumus (2, 11), dan K merupakan banyak urutan berkala akhir.

Untuk mencari banyak urutan berkala akhir, digunakan persamaan (2, 9), dimana N merupakan banyak urutan berkala awal. Persamaan linear akhir untuk N = 2 tertera pada Tabel 2.12. Persamaan linear akhir untuk N = 3 tertera pada Tabel 2.13.

$$W = \frac{K}{2}(P_{terpendek} + P_{terpanjang}) \quad (2, 8)$$

$$K = \left\lceil \frac{N}{2} \right\rceil \quad (2, 9)$$

$$P_{terpendek} = \left\lfloor \frac{N}{2} \right\rfloor + 1 \quad (2, 10)$$

$$P_{terpanjang} = N \quad (2, 11)$$

Tabel 2.12 Persamaan Linear Akhir untuk N=2

x	**Persamaan Linear $f(x)$**
0	$1b'_0 + 0b'_1 = 5$
1	$0b'_0 + 1b'_1 = 7$

Tabel 2.13 Persamaan Linear Akhir untuk N=3

x	**Persamaan Linear $f(x)$**
0	$1c_0 + 0c_1 + 0c_2 + 1b'_0 + 0b'_1 = 15$
1	$0c_0 + 1c_1 + 0c_2 + 0b'_0 + 1b'_1 = 3$
2	$0c_0 + 0c_1 + 1c_2 + 1b'_0 + 0b'_1 = 17$
3	$1c_0 + 0c_1 + 0c_2 + 0b'_0 + 1b'_1 = 2$
4	$0c_0 + 1c_1 + 0c_2 + 1b'_0 + 0b'_1 = 16$

Dari persamaan linear yang tertera pada Tabel 2.12 dan Tabel 2.13, maka selanjutnya dicari solusi dari sistem persamaan linear menggunakan eliminasi gauss dengan kompleksitas $O(N^6)$.

Setelah semua variabel diketahui nilainya, maka semua nilai $f(x)$ dapat dicari dengan menjumlahkan satu demi satu variabel pada

urutan berkala yang sesuai dengan posisi x, meskipun nilai x yang diberikan untuk diketahui nilai $f(x)$-nya hanya sebanyak N^2, dengan kompleksitas $O(N^3)$.

Kompleksitas total solusi penyelesaian yaitu $O(TN^6 + TN^3)$ dimana T merupakan banyak sub-kasus uji dan N merupakan banyak urutan berkala awal.

2.6 Pembuatan Data *Generator*

Data *generator* yang akan dibuat terdiri dari 2 buah data *generator* dengan skenario yang berbeda. Data *generator* skenario 1 dibuat untuk uji coba kebenaran dan uji coba kinerja 20 kali percobaan data masukan acak. Data *generator* skenario 2 dibuat untuk uji coba kinerja pengaruh banyak urutan berkala awal N terhadap pertumbuhan waktu berjalan perangkat lunak hasil implementasi.

Pembuatan data *generator* skenario 1 dilakukan untuk membuat data masukan acak serta membuat data keluaran benar sesuai dengan data masukan acak yang telah dibuat. Data keluaran benar digunakan sebagai pembanding data keluaran perangkat lunak hasil implementasi pada uji coba kebenaran. Data *generator* skenario 1 yang dibuat disesuaikan dengan format masukan dan format keluaran yang telah dijelaskan pada subbab 2.2.

Data *generator* skenario 1 akan membuat data masukan acak dimana banyak sub-kasus uji pada satu data masukan acak berkisaran antara 1 hingga 1024 sesuai dengan batasan jumlah sub-kasus uji. Selanjutnya banyak urutan berkala awal N diberikan sedemikian hingga sesuai dengan batasan, yaitu antara 1 hingga 14. Selanjutnya akan diberikan N^2 nilai $f(x)$ dari $f(0)$ hingga $f(N^2 - 1)$ dengan menyesuaikan penjumlahan N buah urutan berkala, dimana isi dari N buah urutan berkala dibuat secara acak terlebih dahulu. Nilai $f(x)$ berkisar antara -1.000.000.000 hingga

1.000.000.000. Berikutnya, diberikan N^2 buah nilai x, dimana ditanyakan nilai $f(x)$. Nilai x berkisar antara 0 hingga 1.000.000.000.

Selanjutnya, data *generator* skenario 1 akan membuat data keluaran yang sepasang dengan data masukan acak yang telah dibuat. Data keluaran acak dibuat berdasarkan N buah urutan berkala yang telah dibuat isinya ketika membuat data masukan acak. Isi dari N urutan berkala diiterasi dan ditambah satu persatu secara manual sehingga menghasilkan data keluaran benar. Data keluaran benar dipastikan benar karena hasil dibuat dari menambahkan secara manual N buah urutan berkala yang telah diketahui isinya. Nilai keluaran $f(x)$ berkisar antara -10^9 hingga 10^9.

Data *generator* skenario 2 merupakan modifikasi dari data *generator* skenario 1. Perbedaannya hanya terdapat pada jumlah sub-kasus uji yang diberikan dan masukan banyak urutan berkala awal N. Pada data *generator* skenario 2, jumlah sub-kasus uji dibatasi sedemikian hingga selalu menghasilkan 1 buah sub-kasus uji saja sehingga terlihat kinerja untuk banyak urutan berkala N. Banyak urutan berkala awal N bukan merupakan bilangan acak, melainkan dimasukkan sendiri oleh pengguna dan tetap mengikuti batasan nilai N, yaitu antara 1 hingga 14.

2.7 Pembuatan Penguji Coba Kebenaran

Pembuatan penguji coba kebenaran dibuat untuk membandingkan data keluaran perangkat lunak yang telah diimplementasi dengan data keluaran benar. Jika semua data keluaran solusi sesuai dengan data keluaran benar untuk semua sub-kasus uji, maka solusi penyelesaian permasalahan dapat dikatakan benar. Jika ada bagian dari data keluaran solusi yang berbeda dengan data keluaran benar, maka solusi masih belum dapat memprediksi dengan tepat nilai

$f(x)$. Jika hal ini terjadi, maka akan dikeluarkan hasil berupa total selisih perbedaan hasil solusi dan hasil benar, serta banyak nilai $f(x)$ keluaran solusi yang sama dengan $f(x)$ keluaran benar per banyak $f(x)$ keluaran benar. Jika total selisih perbedaan hasil solusi dan hasil benar bernilai 0 serta menghasilkan akurasi 100%, maka solusi dikatakan benar.

(Halaman ini sengaja dikosongkan)

BAB 3
DESAIN

Pada bab ini dijelaskan desain algoritma yang digunakan dalam menyelesaikan permasalahan pada Tugas Akhir ini.

3.1 Definisi Umum Sistem

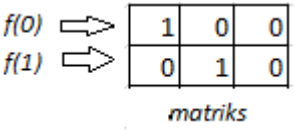
Sistem pertama kali akan menjalankan fungsi MAIN terlebih dahulu. Desain fungsi MAIN tertera pada Gambar 3.2. Di dalam fungsi MAIN, dipanggil fungsi INIT-MATRIKS yang bertugas untuk mengisi nilai matriks awal yang akan digunakan untuk eliminasi gauss. Setelah itu, terdapat fungsi GAUSS-ELIMINASI yang bertugas menjalankan proses eliminasi gauss dari matriks yang telah diinisiasi sebelumnya. Setelah proses eliminasi gauss selesai dan mendapatkan nilai-nilai variabel, maka selanjutnya dilakukan pemanggilan fungsi QUERY yang bertugas untuk mencari dan mencetak nilai $f(x)$ yang ditanyakan.

3.2 Desain Algoritma

Sistem terdiri dari beberapa fungsi, yaitu INIT-MATRIKS, GAUSS-ELIMINASI, dan QUERY. Pada subbab ini dijelaskan desain algoritma dari masing-masing fungsi tersebut.

3.2.1 Desain Fungsi INIT-MATRIKS

Fungsi INIT-MATRIKS berguna untuk menginisiasi variabel $matriks$ bertipe data larik dua dimensi dengan nilai 0 dan 1. Sebagai contoh, untuk kasus banyak urutan berkala awal $N = 2$, hasil inisiasi variabel $matriks$ tertera pada Gambar 3.1. Untuk kasus banyak urutan berkala awal $N = 3$, hasil inisiasi variabel $matriks$ tertera pada Gambar 3.3. Untuk kasus banyak urutan berkala awal $N = 4$, hasil inisiasi variabel $matriks$ tertera pada

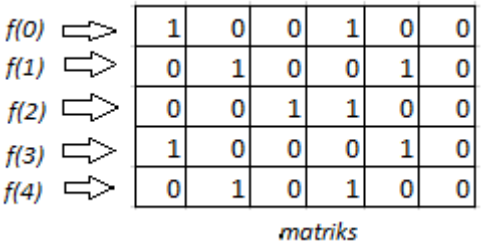


Gambar 3.1 Hasil Inisiasi Variabel matriks untuk N=2

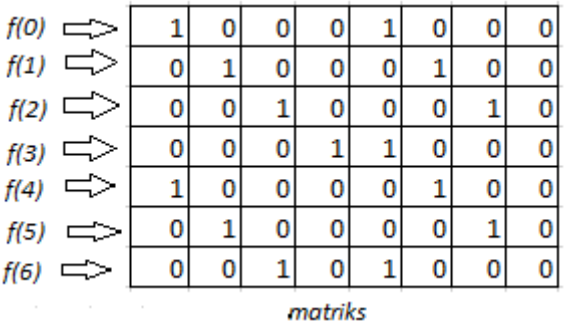
```
MAIN()
 1  input t
 2  for j=1 to t
 3     input n
 4     awal = (n/2)+1
 5     akhir= n
 6     banyak_period = max(n/2,n-(n/2))
 7     banyak_koef = banyak_period*(awal+akhir)/2
 8     n_kuadrat = n*n
 9     let matriks[0..(banyak_koef-
    1)][0..banyak_koef] be a new array
10     let koef[0..(banyak_koef-1)] be a new array
11     matriks = INIT-MATRIKS(matriks, banyak_koef, n)
12     for i=0 to banyak_koef-1
13        input matriks[i][banyak_koef]
14     for i=banyak_koef to n_kuadrat-1
15        input y
16     koef = GAUSS-ELIMINASI(matriks, koef,
    banyak_koef)

17     QUERY(koef, n_kuadrat, n, banyak_koef)
18     print "\n"
19  return
```

Gambar 3.2 Pseudocode Fungsi MAIN

Gambar 3.4. Setelah variabel *matriks* diinisiasi, isi variabel *matriks* menjadi nilai kembalian dari fungsi INIT-MATRIKS. Desain fungsi INIT-MATRIKS tertera pada Gambar 3.5.



Gambar 3.3 Hasil Inisiasi Variabel matriks untuk N=3



Gambar 3.4 Hasil Inisiasi Variabel matriks untuk N=4

```
INIT-MATRIKS(matriks, banyak_koef, n)
 1  for i=0 to banyak_koef-1
 2     for j=0 to banyak_koef
 3        matriks[i][j] = 0
 4  period_now = n+1
 5  for k=0 to banyak_koef-1 increment by
    period_now
 6     period_now = period_now - 1
 7     index = 0
 8     for i=0 to banyak_koef-1
 9        matriks[i][k+index] = 1
10        index = index + 1
11        if index = period_now
12           index = 0
13  return matriks
```

Gambar 3.5 Pseudocode Fungsi INIT-MATRIKS

3.2.2 Desain Fungsi GAUSS-ELIMINASI

Fungsi GAUSS-ELIMINASI digunakan untuk melaksanakan proses eliminasi gauss dari matriks yang telah diinisiasi dengan nilai 0 dan 1 serta telah diisi dengan nilai $f(0)$ hingga $f(W-1)$ pada kolom terakhir variabel $matriks$. Hasil solusi sistem persamaan linear, dimana merupakan elemen-elemen dari N urutan berkala, disimpan ke dalam variabel $koef$. Isi dari variabel $koef$ menjadi nilai kembalian dari fungsi GAUSS-ELIMINASI. Desain fungsi GAUSS-ELIMINASI tertera pada Gambar 3.7.

3.2.3 Desain Fungsi QUERY

Fungsi QUERY digunakan untuk mencetak nilai $f(x)$ yang ditanyakan berdasarkan nilai x yang dimasukkan. Desain fungsi QUERY tertera pada Gambar 3.6.

```
QUERY(koef, n_kuadrat, n, banyak_koef)
 1  for q=0 to n_kuadrat-1
 2     input x
 3     res = 0
 4     period_now = n+1
 5     for padding=0 to banyak_koef-1 increment by
    period_now
 6        period_now = period_now - 1
 7        res = res + koef[padding+(x mod
    period_now)]
 8     if q>0
 9        print " "
10     print res
11  return
```

Gambar 3.6 Pseudocode Fungsi QUERY

3.3 Desain Data *Generator*

Data *generator* skenario 1 digunakan untuk membuat kasus uji coba acak sesuai dengan format masukan dan membuat keluaran

```
GAUSS-ELIMINASI(matriks, koef, banyak_koef)
 1  for i=0 to banyak_koef-2
 2     if matriks[i][i] = 0
 3        status = false
 4        for j=i+1 to banyak_koef-1
 5           if matriks[j][i] != 0
 6              for k=i to banyak_koef
 7                 temp = matriks[j][k]
 8                 matriks[j][k] = matriks[i][k]
 9                 matriks[i][k] = temp
10              status = true
11              break
12        if status = false
13           continue
14     for j=i+1 to banyak_koef-1
15        if matriks[j][i] = 0
16           continue
17        fpb = gcd(abs(matriks[j][i]),
    abs(matriks[i][i]))
18        pengali_atas = matriks[j][i]/fpb
19        pengali_bawah = matriks[i][i]/fpb
20        for k=i to banyak_koef
21           matriks[j][k] =
    matriks[j][k]*pengali_bawah -
    matriks[i][k]*pengali_atas
22  for i=banyak_koef-1 to 0
23     hasil = matriks[i][banyak_koef]
24     for j=banyak_koef-1 to i+1
25        hasil = hasil - matriks[i][j]*koef[j]
26     if matriks[i][i] != 0
27        koef[i] = hasil/matriks[i][i]
28     else
29        koef[i] = 3  //random
30  return koef
```

Gambar 3.7 Pseudocode Fungsi GAUSS-ELIMINASI

benar dari kasus uji coba acak tersebut. Desain data *generator* skenario 1 tertera pada Gambar 3.8 dan Gambar 3.9.

```
DATA-GENERATOR-1()
 1  t = (random() mod 1024) + 1
 2  print t, "\n" to input file
 3  while t>0
 4     t = t - 1
 5     n = (random() mod 14) + 1
 6     n_kuadrat = n*n
 7     print n, "\n" to input file
 8     let sequence[0..(n-1)][0..(n-1)] new array
 9     for i=0 to n-1
10        for j=0 to i
11  sequence[i][j]= random() mod ((1000000000/n)+1)
12           tanda = random() mod 2
13           if tanda = 1
14              sequence[i][j] = sequence[i][j]*-1
15     for i=0 to n_kuadrat-1
16        y = 0
17        for j=0 to n-1
18           y = y + sequence[j][i mod (j+1)]
19        if i = 0
20           print y to input file
21        else
22           print " ", y to input file
23     print "\n" to input file
24     for i=0 to n_kuadrat-1
25        x = random() mod 1000000001
26        if i = 0
27           print x to input file
28        else
29           print " ", x to input file
30        y = 0
31        for k=0 to n-1
32           y = y + sequence[k][x mod (k+1)]
```

Gambar 3.8 Pseudocode Data Generator Skenario 1 (1)

```
DATA-GENERATOR-1()
33          if i = 0
34             print y to output file
35          else
36             print " ", y to output file
37       print "\n" to input file
38       print “\n” to output file
39    return
```

Gambar 3.9 Pseudocode Data Generator Skenario 1 (2)

```
PENGUJI-COBA-KEBENARAN()
 1  delta = 0
 2  akurasi = 0
 3  all = 0
 4  while (hasil_solusi = scan integer output from
    program in file)
 5     hasil_benar = scan integer real output in
    file
 6     if hasil_solusi = hasil_benar
 7        akurasi = akurasi + 1
 8     else
 9        if hasil_solusi > hasil_benar
10           delta = delta + (hasil_solusi -
    hasil_benar)
11        else
12           delta = delta + (hasil_benar -
    hasil_solusi)
13     all = all + 1
14  print "delta: ", delta, "\n"
15  print "akurasi: ", akurasi, "/", all, "\n"
```

Gambar 3.10 Pseudocode Penguji Coba Kebenaran

Data *generator* skenario 2 digunakan untuk membuat data masukan untuk mengukur pengaruh banyak urutan berkala N terhadap pertumbuhan waktu berjalan perangkat lunak hasil

```
DATA-GENERATOR-2()
 1  t = 1
 2  print t, "\n"
 3  while t>0
 4     t = t - 1
 5     input n
 6     n_kuadrat = n*n
 7     print n, "\n"
 8     let sequence[0..(n-1)][0..(n-1)] new array
 9     for i=0 to n-1
10        for j=0 to i
11           sequence[i][j] = random() mod
    ((1000000000/n)+1)
12           tanda = random() mod 2
13           if tanda = 1
14              sequence[i][j] = sequence[i][j]*-1
15     for i=0 to n_kuadrat-1
16        y = 0
17        for j=0 to n-1
18           y = y + sequence[j][i mod (j+1)]
19        if i = 0
20           print y
21        else
22           print " ", y
23     print "\n"
24     for i=0 to n_kuadrat-1
25        x = random() mod 1000000001
26        if i = 0
27           print x
28        else
29           print " ", x
30     print "\n"
31  return
```

Gambar 3.11 Pseudocode Data Generator Skenario 2

implementasi. Desain data *generator* skenario 2 tertera pada Gambar 3.11.

3.4 Desain Penguji Coba Kebenaran

Penguji coba kebenaran digunakan untuk menguji hasil keluaran solusi yang dibandingkan dengan hasil keluaran sebenarnya. Desain penguji coba kebenaran tertera pada Gambar 3.10. Variabel $delta$ digunakan untuk menampung jumlah selisih perbedaan hasil $f(x)$ solusi dan $f(x)$ sebenarnya. Variabel $akurasi$ digunakan untuk menghitung banyak hasil $f(x)$ solusi yang sama dengan hasil $f(x)$ sebenarnya. Variabel all digunakan untuk menghitung banyak nilai $f(x)$ sebenarnya pada satu berkas.

(Halaman ini sengaja dikosongkan)

BAB 4
IMPLEMENTASI

Pada bab ini dijelaskan implementasi sesuai dengan desain algoritma yang telah ditentukan sebelumnya.

4.1 Lingkungan Implementasi

Lingkungan uji coba yang digunakan adalah sebagai berikut:

1. Perangkat Keras:
   Prosesor: Intel® Core™ i5 2430M @ 2.40 GHz x 2
   RAM: 4.00 GB
2. Perangkat Lunak:
   Sistem Operasi: Windows 8.1 64 -bit
   IDE: Bloodshed Dev-C++ 5.11
   Compiler : g++ (TDM-GCC 4.9.2 32-bit)

4.2 Penggunaan *Library*, Konstanta, dan Variabel Global

Pada subbab ini akan dibahas penggunaan *library*, konstanta dan variabel global yang digunakan dalam sistem. Potongan kode yang menyatakan penggunaan *library* ditunjukkan pada Kode Sumber 4.1.

```
 1  #include <stdio.h>
 2  #include <vector>
 3  #include <cstdlib>
 4  #include <iostream>
 5  #include <iomanip>
 6  #include <string>
 7  #include <time.h>
 8  using namespace std;
 9  const int base = 1000000000;
10  const int base_digits = 9;
```

Kode Sumber 4.1 Potongan Kode Penggunaan Library

Pada Kode Sumber 4.1, terdapat tujuh *library* yang digunakan yaitu *stdio.h*, *vector*, *cstdlib*, *iostream*, *iomanip*, *string*, dan *time.h*. Terdapat sebuah kelas *bigint* yang berguna untuk menampung bilangan bulat dengan jumlah digit lebih dari 19 digit. Kelas *bigint* tidak dicantumkan di sini karena hak cipta kode sumber kelas bukanlah milik penulis. Kode sumber kelas *bigint* diambil dari https://pastebin.com/aQ8NJ197. Konstanta *base* dan *base_digits* pada Kode Sumber 4.1 merupakan konstanta yang dibutuhkan oleh kelas *bigint*, dimana kedua konstanta ini juga diambil dari https://pastebin.com/aQ8NJ197.

4.3 Implementasi Fungsi MAIN

Fungsi MAIN diimplementasikan sesuai dengan desain fungsi MAIN pada Gambar 3.2 yang ditunjukkan pada Kode Sumber 4.3. Variabel global yang dibutuhkan oleh fungsi MAIN tertera pada Kode Sumber 4.2. Variabel $koef$ digunakan untuk menampung semua angka pada N urutan berkala. Variabel $matriks$ digunakan saat proses eliminasi gauss, dimana membutuhkan larik dua dimensi.

```
1  AccDouble koef[200];
2  bigint matriks[200][201];
```

Kode Sumber 4.2 Potongan Kode Deklarasi Variabel Global

4.4 Implementasi Class $AccDouble$

Class $AccDouble$ merupakan kelas yang menampung bilangan decimal besar dan akurat tanpa kehilangan presisi sedikitpun dengan menyimpan pembilang dan penyebut dari bilangan desimal. Pembilang dan penyebut juga dapat menyimpan bilangan bulat besar, karena pembilang dan penyebut dalam class ini menggunakan sub-class *bigint*. Implementasi class $AccDouble$ ditunjukkan pada . Di dalamnya, terdapat konstruktor, operator

```
 1  int main(){
 2     int t;
 3     int n;
 4     scanf ("%d", &t);
 5     while (t--){
 6        scanf ("%d",&n);
 7        int awal = (n/2)+1;
 8        int akhir= n;
 9        int banyak_period = max(n/2,n-(n/2));
10        int banyak_koef =
    banyak_period*(awal+akhir)/2;
11        int n_kuadrat = n*n;
12        initMatriks(banyak_koef, n);
13        for (int i=0;i<banyak_koef;i++){
14           long long y;
15           scanf ("%lld", &y);
16           matriks[i][banyak_koef] = y;
17        }
18        for (int
    i=banyak_koef;i<n_kuadrat;i++){
19           int y;
20           scanf ("%d", &y);
21        }
22        gaussEliminasi(banyak_koef);
23        query(n_kuadrat, n, banyak_koef);
24        printf ("\n");
25     }
26     return 0;
27  }
```

Kode Sumber 4.3 Potongan Kode Fungsi MAIN

tambah, operator kurang, operator kali, dan operator bagi, *method* PRINT-STR() untuk mencetak pembilang, dan *method* NORMALIZE() untuk menyederhanakan pecahan pada class *AccDouble*.

```
 1  class AccDouble{
 2     public:
 3        bigint pembilang;
 4        bigint penyebut;
 5        AccDouble();
 6        AccDouble(int xx);
 7        AccDouble(bigint &xx);
 8        void normalize();
 9        AccDouble operator*(const AccDouble
    &other);
10        AccDouble operator-(const AccDouble
    &other);
11        AccDouble operator/(const AccDouble
    &other);
12        AccDouble operator+(const AccDouble
    &other);
13        void printStr();
14  };
```

Kode Sumber 4.4 Potongan Kode Class AccDouble

4.4.1 Implementasi Konstruktor AccDouble

Implementasi konstruktor pada class *AccDouble* pada Kode Sumber 4.4 baris kelima, keenam, dan ketujuh, terdapat pada Kode Sumber 4.5.

```
 1  AccDouble(){
 2     this->pembilang = 0LL;
 3     this->penyebut = 1LL;
 4  }
 5  AccDouble(int xx){
 6     this->pembilang = (long long)xx;
 7     this->penyebut = 1LL;
 8  }
 9  AccDouble(bigint &xx){
10     this->pembilang = xx;
11     this->penyebut = 1LL;
12  }
```

Kode Sumber 4.5 Potongan Kode Konstruktor AccDouble

4.4.2 Implementasi Method NORMALIZE

Implementasi method NORMALIZE pada class *AccDouble* pada Kode Sumber 4.4 baris kedelapan terdapat pada Kode Sumber 4.6.

```
1  void normalize(){
2     if (penyebut.sign==-1){
3        penyebut.sign = 1;
4        pembilang.sign *= -1;
5     }
6     bigint fpb = gcd(pembilang.abs(), penyebut);
7     pembilang /= fpb;
8     penyebut /= fpb;
9  }
```

Kode Sumber 4.6 Potongan Kode Method NORMALIZE

4.4.3 Implementasi Operator *

Implementasi operator * pada class *AccDouble* pada Kode Sumber 4.4 baris kesembilan terdapat pada Kode Sumber 4.7.

```
1  AccDouble operator*(const AccDouble &other){
2     AccDouble res(0);
3     res.pembilang = this->pembilang *
   other.pembilang;
4     res.penyebut = this->penyebut *
   other.penyebut;
5     res.normalize();
6     return res;
7  }
```

*Kode Sumber 4.7 Potongan Kode Operator **

4.4.4 Implementasi Operator -

Implementasi operator - pada class *AccDouble* pada Kode Sumber 4.4 baris kesepuluh terdapat pada Kode Sumber 4.8.

```
 1  AccDouble operator-(const AccDouble &other){
 2     AccDouble res(0);
 3     bigint fpb = gcd(this->penyebut,
    other.penyebut);
 4     bigint pengali_this = other.penyebut/fpb;
 5     bigint pengali_other = this->penyebut/fpb;
 6     bigint this_pembilang = this->pembilang *
    pengali_this;
 7     bigint this_penyebut = this->penyebut *
    pengali_this;
 8     bigint other_pembilang = other.pembilang *
    pengali_other;
 9     res.pembilang = this_pembilang -
    other_pembilang;
10     res.penyebut = this_penyebut;
11     res.normalize();
12     return res;
13  }
```

Kode Sumber 4.8 Potongan Kode Operator –

4.4.5 Implementasi Operator /

Implementasi operator / pada class *AccDouble* pada Kode Sumber 4.4 baris kesebelas terdapat pada Kode Sumber 4.9.

```
 1  AccDouble operator/(const AccDouble &other){
 2     AccDouble res(0);
 3     res.pembilang = this->pembilang *
    other.penyebut;
 4     res.penyebut = this->penyebut *
    other.pembilang;
 5     res.normalize();
 6     return res;
 7  }
```

Kode Sumber 4.9 Potongan Kode Operator /

4.4.6 Implementasi Operator +

Implementasi operator + pada class *AccDouble* pada Kode Sumber 4.4 baris keduabelas terdapat pada Kode Sumber 4.10.

```
 1  AccDouble operator+(const AccDouble &other){
 2     AccDouble res(0);
 3     bigint fpb = gcd(this->penyebut,
    other.penyebut);
 4     bigint pengali_this = other.penyebut/fpb;
 5     bigint pengali_other = this->penyebut/fpb;
 6     bigint this_pembilang = this->pembilang *
    pengali_this;
 7     bigint this_penyebut = this->penyebut *
    pengali_this;
 8     bigint other_pembilang = other.pembilang *
    pengali_other;
 9     res.pembilang = this_pembilang +
    other_pembilang;
10     res.penyebut = this_penyebut;
11     res.normalize();
12     return res;
13  }
```

Kode Sumber 4.10 Potongan Kode Operator +

4.4.7 Implementasi Method PRINT-STR

Implementasi method PRINT-STR pada class *AccDouble* pada Kode Sumber 4.4 baris ketigabelas terdapat pada Kode Sumber 4.11.

```
 1  void printStr(){
 2     cout << pembilang;
 7  }
```

Kode Sumber 4.11 Potongan Kode Method PRINT-STR

4.5 Implementasi Fungsi INIT-MATRIKS

Implementasi fungsi INIT-MATRIKS pada Kode Sumber 4.12 mengikuti desain fungsi yang tertera pada Gambar 3.5.

```
 1  void initMatriks(int banyak_koef, int n){
 2     for (int i=0;i<banyak_koef;i++){
 3        for (int j=0;j<=banyak_koef;j++){
 4           matriks[i][j].a.clear();
 5           matriks[i][j].sign = 1;
 6        }
 7     }
 8     int period_now = n+1;
 9     for (int k=0;k<banyak_koef;k+=period_now){
10        period_now--;
11        int index = 0;
12        for (int i=0;i<banyak_koef;i++){
13           matriks[i][k+index] = 1LL;
14           index++;
15           if (index==period_now) index = 0;
16        }
17     }
18  }
```

Kode Sumber 4.12 Potongan Kode Fungsi INIT-MATRIKS

4.6 Implementasi Fungsi GAUSS-ELIMINASI

Implementasi fungsi GAUSS-ELIMINASI pada Kode Sumber 4.13 dan Kode Sumber 4.14 mengikuti desain fungsi yang tertera pada Gambar 3.7.

4.7 Implementasi Fungsi QUERY

Implementasi fungsi QUERY pada Kode Sumber 4.15 mengikuti desain fungsi yang tertera pada Gambar 3.4.

```
 1  void gaussEliminasi(int banyak_koef){
 2     for (int i=0;i<banyak_koef-1;i++){
 3        if (matriks[i][i].isZero()){
 4           bool status = false;
 5           for (int j=i+1;j<banyak_koef;j++){
 6              if (!matriks[j][i].isZero()){
 7                 for (int
    k=i;k<=banyak_koef;k++){
 8                    bigint temp;
 9                    temp = matriks[j][k];
10                    matriks[j][k] =
    matriks[i][k];
11                    matriks[i][k] = temp;
12                 }
13                 status = true;
14                 break;
15              }
16           }
17           if (status==false) continue;
18        }
19        for (int j=i+1;j<banyak_koef;j++){
20           if (matriks[j][i].isZero())
    continue;
21           bigint fpb =
    gcd(matriks[j][i].abs(),
    matriks[i][i].abs());
22           bigint pengali_atas =
    matriks[j][i]/fpb;
23           bigint pengali_bawah =
    matriks[i][i]/fpb;
24           for (int k=i;k<=banyak_koef;k++){
25              matriks[j][k] =
    matriks[j][k]*pengali_bawah -
    matriks[i][k]*pengali_atas;
26           }
27        }
28     }
```

Kode Sumber 4.13 Potongan Kode Fungsi GAUSS-ELIMINASI (1)

```
29      for (int i=banyak_koef-1;i>=0;i--){
30         AccDouble
    hasil(matriks[i][banyak_koef]);
31         for (int j=banyak_koef-1;j>i;j--){
32            AccDouble temp(matriks[i][j]);
33            hasil = hasil - temp*koef[j];
34         }
35         if (!matriks[i][i].isZero()){
36            AccDouble temp(matriks[i][i]);
37            koef[i] = hasil/temp;
38         }
39         else{
40            koef[i] = AccDouble(3); //random
41         }
42      }
43      return;
44   }
```

Kode Sumber 4.14 Potongan Kode Fungsi GAUSS-ELIMINASI (2)

```
 1   void query(int n_kuadrat, int n, int
    banyak_koef){
 2      for (int q=0;q<n_kuadrat;q++){
 3         int x;
 4         scanf ("%d", &x);
 5         AccDouble res(0);
 6         int period_now = n+1;
 7         for (int
    pad=0;pad<banyak_koef;pad+=period_now){
 8            period_now--;
 9            res = res +
    koef[pad+(x%period_now)];
10         }
11         if (q>0) printf(" ");
12         res.printStr();
13      }
14   }
```

Kode Sumber 4.15 Potongan Kode Fungsi QUERY

4.8 Implementasi Data *Generator*

Implementasi pada Kode Sumber 4.16 dan Kode Sumber 4.17 mengikuti desain data *generator* skenario 1 yang tertera pada Gambar 3.8 dan Gambar 3.9.

```
 1  #include<stdio.h>
 2  #include<stdlib.h>
 3  #include<time.h>
 4  int main(){
 5     srand(time(NULL));
 6     FILE * pIn;
 7     FILE * pOut;
 8     char nama_masuk[10], nama_keluar[10];
 9     scanf ("%s %s",nama_masuk, nama_keluar);
10     pIn = fopen(nama_masuk,"w+");
11     pOut = fopen(nama_keluar,"w+");
12     int t = (rand() % 1024) + 1;
13     fprintf (pIn, "%d\n", t);
14     while (t--){
15        int n = (rand() % 14) + 1;
16        int n_kuadrat = n*n;
17        fprintf (pIn, "%d\n",n);
18        int sequence[n][n];
19        for (int i=0; i<n; i++){
20           for (int j=0; j<=i; j++){
21              sequence[i][j] = rand() %
    ((1000000000/n)+1);
22              int tanda = rand() % 2;
23              if (tanda==1) sequence[i][j] *= -1;
24           }
25        }
26        for (int i=0;i<n_kuadrat;i++){
27           int y = 0;
```

Kode Sumber 4.16 Implementasi Data Generator Skenario 1 (1)

```
28            for (int j=0; j<n; j++){
29               y += sequence[j][i%(j+1)];
30            }
31            if (i==0) fprintf (pIn,"%d",y);
32            else fprintf (pIn," %d",y);
33         }
34         fprintf (pIn,"\n");
35         for (int i=0;i<n_kuadrat;i++){
36            int x = rand() % 1000000001;
37            if (i==0) fprintf (pIn,"%d",x);
38            else fprintf (pIn," %d",x);
39            int y = 0;
40            for (int k=0; k<n; k++){
41               y += sequence[k][x%(k+1)];
42            }
43            if (i==0) fprintf (pOut, "%d",y);
44            else fprintf (pOut, " %d",y);
45         }
46         fprintf (pIn,"\n");
47         fprintf (pOut, "\n");
48      }
49      return 0;
50   }
```

Kode Sumber 4.17 Implementasi Data Generator Skenario 1 (2)

Implementasi pada Kode Sumber 4.18 mengikuti desain data *generator* skenario 2 yang tertera pada Gambar 3.11.

4.9 Desain Penguji Coba Kebenaran

Implementasi pada Kode Sumber 4.19 mengikuti desain penguji coba kebenaran yang tertera pada Gambar 3.10.

```
 1  #include<stdio.h>
 2  #include<stdlib.h>
 3  #include<time.h>
 4  int main(){
 5     srand(time(NULL));
 6     int t = 1;
 7     printf ("%d\n", t);
 8     while (t--){
 9        int n;
10        scanf ("%d", &n);
11        int n_kuadrat = n*n;
12        printf ("%d\n",n);
13        int sequence[n][n];
14        for (int i=0; i<n; i++){
15           for (int j=0; j<=i; j++){
16        sequence[i][j]=rand()%((1000000000/n)+1);
17              int tanda = rand() % 2;
18              if (tanda==1) sequence[i][j] *= -1;
19           }
20        }
21        for (int i=0;i<n_kuadrat;i++){
22           int y = 0;
23           for (int j=0; j<n; j++){
24              y += sequence[j][i%(j+1)];
25           }
26           if (i==0) printf ("%d",y);
27           else printf (" %d",y);
28        }
29        printf ("\n");
30        for (int i=0;i<n_kuadrat;i++){
31           int x = rand() % 1000000001;
32           if (i==0) printf ("%d",x);
33           else printf (" %d",x);
34        }
35        printf ("\n");
36     }
37     return 0;}
```

Kode Sumber 4.18 Implementasi Data Generator Skenario 2

```
 1  #include<stdio.h>
 2  int main(){
 3     FILE * pSolusi;
 4     FILE * pBenar;
 5     char nama_solusi[10], nama_benar[10];
 6     scanf ("%s %s",nama_solusi, nama_benar);
 7     pSolusi = fopen(nama_solusi,"r");
 8     pBenar = fopen(nama_benar,"r");
 9     int hasil_solusi, hasil_benar;
10     int delta = 0, akurasi = 0, all = 0;
11     while
    (fscanf(pSolusi,"%d",&hasil_solusi)!=EOF){
12        fscanf(pBenar,"%d",&hasil_benar);
13        if (hasil_solusi==hasil_benar){
14           akurasi = akurasi + 1;
15        }
16        else{
17           if (hasil_solusi>hasil_benar)
18              delta = delta + (hasil_solusi -
    hasil_benar);
19           else
20              delta = delta + (hasil_benar -
    hasil_solusi);
21        }
22        all = all + 1;
23     }
24     printf ("delta: %d\n",delta);
25     printf ("akurasi: %d/%d\n",akurasi,all);
26     fclose(pSolusi);
27     fclose(pBenar);
28     return 0;
29  }
```

Kode Sumber 4.19 Implementasi Penguji Coba Kebenaran

BAB 5
UJI COBA DAN EVALUASI

Pada bab ini dijelaskan tentang uji coba dan evaluasi dari implementasi yang telah dilakukan pada BAB 4.

5.1 Lingkungan Uji Coba

Lingkungan uji coba yang digunakan adalah komputer penulis dengan spesifikasi sebagai berikut:

3. Perangkat Keras:
   Prosesor: Intel® Core™ i5 2430M @ 2.40 GHz x 2
   RAM: 4.00 GB
4. Perangkat Lunak:
   Sistem Operasi: Windows 8.1 64 -bit
   IDE: Bloodshed Dev-C++ 5.11
   Compiler : g++ (TDM-GCC 4.9.2 32-bit)

5.2 Skenario Uji Coba

Pada subbab ini akan dijelaskan uji coba yang dilakukan. Terdapat dua buah uji coba yang dilakukan, yaitu uji coba kebenaran dan uji coba kinerja.

5.2.1 Uji Coba Kebenaran

Uji coba kebenaran yang dilakukan adalah menganalisis kasus uji coba sederhana dengan langkah-langkah yang telah dijelaskan pada subbab 2.5 untuk memvalidasi kebenaran hasil keluaran program sesuai dengan hasil keluaran yang diharapkan. Uji coba juga dilakukan dengan mencocokan hasil keluaran perangkat lunak berdasarkan data masukan acak yang dibuat oleh data *generator* skenario 1, dengan hasil keluaran sesungguhnya yang dibuat oleh data *generator* skenario 1.

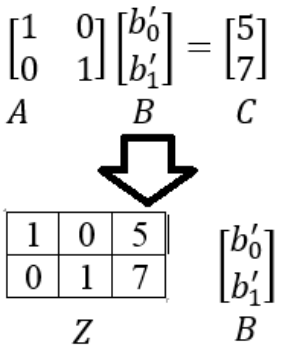
```
1  2
2  2
3  5 7 5 7
4  6 7 8 9
5  3
6  15 3 17 2 16 4 15 3 17
7  10 100 1000 10000 100000 1000000 10000000
   100000000 1000000000
```

Gambar 5.1 Contoh Kasus Uji Coba

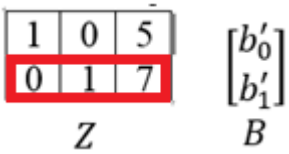
Kasus uji coba yang digunakan sebagai masukkan uji coba sederhana dalam menganalisis uji coba kebenaran adalah data masukan yang terdapat pada soal SPOJ dan pada Gambar 5.1.

Pertama, diberikan 2 sub-kasus uji T dengan sub-kasus uji pertama diberikan banyak urutan berkala awal N sebanyak 2 buah dan sub-kasus uji kedua diberikan banyak urutan berkala awal N sebanyak 3 buah.

Pada sub-kasus uji pertama, diketahui dua buah urutan berkala awal dan nilai dari $f(0)$, $f(1)$, $f(2)$, serta $f(3)$. Untuk mencari banyaknya variabel yang dibutuhkan untuk proses eliminasi gauss, digunakan rumus yang tertera pada rumus (2, 8). Sebelum masuk ke rumus (2, 8), terlebih dahulu dicari nilai K dengan rumus (2, 9), nilai $P_{terpendek}$ dengan rumus (2, 10), dan nilai $P_{terpanjang}$ dengan rumus (2, 11). Dari ketiga rumus sebelumnya, didapat nilai $K = \left\lceil \frac{N}{2} \right\rceil = \left\lceil \frac{2}{2} \right\rceil = 1$, $P_{terpendek} = \left\lfloor \frac{N}{2} \right\rfloor + 1 = \left\lfloor \frac{2}{2} \right\rfloor + 1 = 2$, dan $P_{terpanjang} = N = 2$. Banyak variabel yang dibutuhkan yaitu $W = \frac{K}{2}\left(P_{terpendek} + P_{terpanjang}\right) = \frac{1}{2}(2 + 2) = 2$ variabel. Jumlah persamaan linear yang dibutuhkan mengikuti jumlah variabel yang dibutuhkan, yaitu 2 persamaan linear. Karena hanya membutuhkan 2 persamaan, maka persamaan yang diambil hanya $f(0)$ dan $f(1)$ saja. Selanjutnya akan terbentuk perkalian matriks,



Gambar 5.2 Ilustrasi Pembentukan Matriks Z untuk Sub-Kasus Uji Pertama



Gambar 5.3 Ilustrasi Matriks Z setelah Iterasi 1 Eliminasi Gauss Sub-Kasus Uji Pertama

dimana matriks A dan matriks C dapat disatukan menjadi sebuah matriks Z seperti yang terlihat pada Gambar 5.2.

Pada iterasi 1 eliminasi gauss, bilangan pada baris ke-2 dilakukan operasi baris sehingga nilai bilangan pada baris-2 kolom-1 bernilai 0. Karena bilangan pada baris-2 kolom-1 sudah bernilai 0, maka bilangan pada baris-2 sudah tidak perlu diubah lagi. Isi matriks Z setelah iterasi 1 ditunjukkan pada Gambar 5.3.

Selanjutnya, dilakukan *back substitution* untuk mencari nilai b'_0 dan nilai b'_1. Pada iterasi 1 *back substitution*, didapatkan nilai b'_1 dengan membagi bilangan pada baris-2 kolom-3 dengan bilangan pada baris-2 kolom-2. Ilustrasi pencarian nilai b'_1 dan terisinya matriks B ditunjukkan pada Gambar 5.4.

$$1b_1' = 7 \qquad \begin{bmatrix} b_0' \\ 7 \end{bmatrix}$$
$$b_1' = 7 \qquad B$$

Gambar 5.4 Ilustrasi Matriks B setelah Iterasi 1 Back Substitution Sub-Kasus Uji Pertama

$$1b_0' + 0b_1' = 5 \qquad \begin{bmatrix} 5 \\ 7 \end{bmatrix}$$
$$b_0' = 5 \qquad B$$

Gambar 5.5 Ilustrasi Matriks B Setelah Iterasi 2 Back Substitution Sub-Kasus Uji Pertama

$f(6) = 1b_0' + 0b_1' = 1(5) + 0(7) = 5$
$f(7) = 0b_0' + 1b_1' = 0(5) + 1(7) = 7$
$f(8) = 1b_0' + 0b_1' = 1(5) + 0(7) = 5$
$f(9) = 0b_0' + 1b_1' = 0(5) + 1(7) = 7$

Gambar 5.6 Ilustrasi Pencarian Jawaban Sub-Kasus Uji Coba Pertama

Pada iterasi 2 *back substitution*, didapatkan nilai b_0' dengan membagi bilangan pada baris-1 kolom-3 dengan bilangan pada baris-1 kolom-1. Ilustrasi pencarian nilai b_0' dan terisinya matriks B ditunjukkan pada Gambar 5.5.

Setelah semua nilai variabel pada matriks B didapat, maka nilai $f(x)$ yang ditanyakan, yaitu $f(6)$, $f(7)$, $f(8)$, dan $f(9)$ bisa dicari nilainya. Proses pencarian nilai $f(6)$, $f(7)$, $f(8)$, dan $f(9)$ dapat dilihat pada Gambar 5.6.

Jawaban yang dihasilkan sesuai dengan keluaran kasus uji coba pada sumber asli soal yang tertera pada Gambar 5.7 baris pertama untuk sub-kasus uji pertama.

```
1  5 7 5 7
2  16 16 16 16 16 16 16 16 16
```

Gambar 5.7 Keluaran Contoh Kasus Uji Coba pada Sumber Asli Soal

Pada sub-kasus uji kedua, diketahui tiga buah urutan berkala awal dan nilai dari $f(0), f(1), f(2), f(3), f(4), f(5), f(6), f(7)$, serta $f(8)$. Untuk mencari banyaknya variabel yang dibutuhkan untuk proses eliminasi gauss, digunakan rumus yang tertera pada rumus (2, 8). Sebelum masuk ke rumus (2, 8), terlebih dahulu dicari nilai K dengan rumus (2, 9), nilai $P_{terpendek}$ dengan rumus (2, 10), dan nilai $P_{terpanjang}$ dengan rumus (2, 11). Dari ketiga rumus sebelumnya, didapat nilai $K = \left\lceil \frac{N}{2} \right\rceil = \left\lceil \frac{3}{2} \right\rceil = 2$, $P_{terpendek} = \left\lfloor \frac{N}{2} \right\rfloor + 1 = \left\lfloor \frac{3}{2} \right\rfloor + 1 = 2$, dan $P_{terpanjang} = N = 3$. Banyak variabel yang dibutuhkan yaitu $W = \frac{K}{2}\left(P_{terpendek} + P_{terpanjang}\right) = \frac{2}{2}(2 + 3) = 5$ variabel. Jumlah persamaan linear yang dibutuhkan mengikuti jumlah variabel yang dibutuhkan, yaitu 5 persamaan linear. Karena hanya membutuhkan 5 persamaan, maka persamaan yang diambil hanya $f(0), f(1), f(2), f(3)$, dan $f(4)$ saja. Selanjutnya akan terbentuk perkalian matriks, dimana matriks A dan matriks C dapat disatukan menjadi sebuah matriks Z seperti yang terlihat pada Gambar 5.8.

Pada iterasi 1 eliminasi gauss, bilangan pada baris ke-2 matriks Z dilakukan operasi baris sehingga nilai bilangan pada baris-2 kolom-1 bernilai 0. Karena bilangan pada baris-2 kolom-1 sudah bernilai 0, maka bilangan pada baris-2 sudah tidak perlu diubah lagi. Isi matriks Z setelah iterasi 1 ditunjukkan pada Gambar 5.9.

Pada iterasi 2 eliminasi gauss, bilangan pada baris ke-3 matriks Z dilakukan operasi baris sehingga nilai bilangan pada baris-3 kolom-1 bernilai 0.

$$\underset{A}{\begin{bmatrix}1 & 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0 & 1\\ 0 & 0 & 1 & 1 & 0\\ 1 & 0 & 0 & 0 & 1\\ 0 & 1 & 0 & 1 & 0\end{bmatrix}}\underset{B}{\begin{bmatrix}a_0\\ a_1\\ a_2\\ b_0'\\ b_1'\end{bmatrix}} = \underset{C}{\begin{bmatrix}15\\ 3\\ 17\\ 2\\ 16\end{bmatrix}}$$

$$\underset{Z}{\begin{array}{|cccccc|}\hline 1 & 0 & 0 & 1 & 0 & 15\\ 0 & 1 & 0 & 0 & 1 & 3\\ 0 & 0 & 1 & 1 & 0 & 17\\ 1 & 0 & 0 & 0 & 1 & 2\\ 0 & 1 & 0 & 1 & 0 & 16\\ \hline\end{array}} \quad \underset{B}{\begin{bmatrix}a_0\\ a_1\\ a_2\\ b_0'\\ b_1'\end{bmatrix}}$$

Gambar 5.8 Ilustrasi Pembentukan Matriks Z untuk Sub-Kasus Uji Kedua

$$\underset{Z}{\begin{array}{|cccccc|}\hline 1 & 0 & 0 & 1 & 0 & 15\\ 0 & 1 & 0 & 0 & 1 & 3\\ 0 & 0 & 1 & 1 & 0 & 17\\ 1 & 0 & 0 & 0 & 1 & 2\\ 0 & 1 & 0 & 1 & 0 & 16\\ \hline\end{array}} \quad \underset{B}{\begin{bmatrix}a_0\\ a_1\\ a_2\\ b_0'\\ b_1'\end{bmatrix}}$$

Gambar 5.9 Ilustrasi Matriks Z setelah Iterasi 1 Eliminasi Gauss Sub-Kasus Uji Kedua

Karena bilangan pada baris-3 kolom-1 telah bernilai 0, maka bilangan pada baris-3 sudah tidak perlu diubah lagi. Isi matriks Z setelah iterasi 2 ditunjukkan pada Gambar 5.10.

1	0	0	1	0	15
0	1	0	0	1	3
0	0	1	1	0	17
1	0	0	0	1	2
0	1	0	1	0	16

Z

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ b_0' \\ b_1' \end{bmatrix}$$

B

Gambar 5.10 Ilustrasi Matriks Z setelah Iterasi 2 Eliminasi Gauss Sub-Kasus Uji Kedua

Baris ke-4 lama	1	0	0	0	1	2
Baris ke-1	1	0	0	1	0	15
Baris ke-4 baru	0	0	0	-1	1	-13

Gambar 5.11 Ilustrasi Pengurangan tiap Elemen Baris ke-4 Iterasi 3

Pada iterasi 3 eliminasi gauss, bilangan pada baris ke-4 matriks *Z* dilakukan operasi baris sehingga nilai bilangan pada baris-4 kolom-1 bernilai 0. Dilakukan operasi pengurangan pada setiap elemen baris ke-4 dengan setiap elemen baris ke-1 dengan kolom yang sama. Ilustrasi pengurangan tiap elemen baris ke-4 tertera pada Gambar 5.11. Isi matriks *Z* setelah iterasi 3 ditunjukkan pada Gambar 5.12.

Pada iterasi 4 eliminasi gauss, bilangan pada baris ke-5 matriks *Z* dilakukan operasi baris sehingga nilai bilangan pada baris-5 kolom-1 bernilai 0. Karena bilangan pada baris-5 kolom-1 sudah bernilai 0, maka bilangan pada baris-5 sudah tidak perlu diubah lagi. Isi matriks *Z* setelah iterasi 4 ditunjukkan pada Gambar 5.13.

1	0	0	1	0	15
0	1	0	0	1	3
0	0	1	1	0	17
0	0	0	-1	1	-13
0	1	0	1	0	16

Z

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ b_0' \\ b_1' \end{bmatrix}$$

B

Gambar 5.12 Ilustrasi Matriks Z setelah Iterasi 3 Eliminasi Gauss Sub-Kasus Uji Kedua

1	0	0	1	0	15
0	1	0	0	1	3
0	0	1	1	0	17
0	0	0	-1	1	-13
0	1	0	1	0	16

Z

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ b_0' \\ b_1' \end{bmatrix}$$

B

Gambar 5.13 Ilustrasi Matriks Z setelah Iterasi 4 Eliminasi Gauss Sub-Kasus Uji Kedua

Pada iterasi 5 eliminasi gauss, bilangan pada baris ke-3 matriks Z dilakukan operasi baris sehingga nilai bilangan pada baris-3 kolom-2 bernilai 0. Karena bilangan pada baris-3 kolom-2 sudah bernilai 0, maka bilangan pada baris-3 sudah tidak perlu diubah lagi. Isi matriks Z setelah iterasi 5 ditunjukkan pada Gambar 5.14.

Pada iterasi 6 eliminasi gauss, bilangan pada baris ke-4 matriks Z dilakukan operasi baris sehingga nilai bilangan pada baris-4 kolom-2 bernilai 0. Karena bilangan pada baris-4 kolom-2 sudah bernilai 0, maka bilangan pada baris ke-4 sudah tidak perlu diubah lagi. Isi matriks Z setelah iterasi 5 ditunjukkan pada Gambar 5.15.

1	0	0	1	0	15
0	1	0	0	1	3
0	0	1	1	0	17
0	0	0	-1	1	-13
0	1	0	1	0	16

Z

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ b'_0 \\ b'_1 \end{bmatrix}$$

B

Gambar 5.14 Ilustrasi Matriks Z setelah Iterasi 5 Eliminasi Gauss Sub-Kasus Uji Kedua

1	0	0	1	0	15
0	1	0	0	1	3
0	0	1	1	0	17
0	0	0	-1	1	-13
0	1	0	1	0	16

Z

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ b'_0 \\ b'_1 \end{bmatrix}$$

B

Gambar 5.15 Ilustrasi Matriks Z setelah Iterasi 6 Eliminasi Gauss Sub-Kasus Uji Kedua

Baris ke-5 lama	0	1	0	1	0	16
Baris ke-2	0	1	0	0	1	3
Baris ke-5 baru	0	0	0	1	-1	13

Gambar 5.16 Ilustrasi Pengurangan tiap Elemen Baris ke-5 Iterasi 7

Pada iterasi 7 eliminasi gauss, bilangan pada baris ke-5 matriks *Z* dilakukan operasi baris sehingga nilai bilangan pada baris-5 kolom-2 bernilai 0. Dilakukan operasi pengurangan pada setiap elemen baris ke-5 dengan setiap elemen baris ke-2 dengan kolom yang sama. Ilustrasi pengurangan tiap elemen baris ke-5 tertera pada Gambar 5.16. Isi matriks *Z* setelah iterasi 7 ditunjukkan pada Gambar 5.17.

1	0	0	1	0	15
0	1	0	0	1	3
0	0	1	1	0	17
0	0	0	-1	1	-13
0	0	0	1	-1	13

Z

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ b'_0 \\ b'_1 \end{bmatrix}$$

B

Gambar 5.17 Ilustrasi Matriks Z setelah Iterasi 7 Eliminasi Gauss Sub-Kasus Uji Kedua

1	0	0	1	0	15
0	1	0	0	1	3
0	0	1	1	0	17
0	0	0	-1	1	-13
0	0	0	1	-1	13

Z

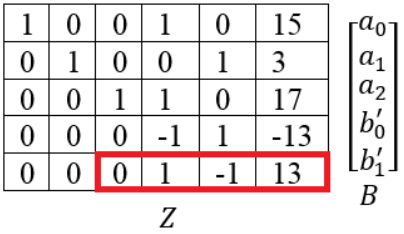
$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ b'_0 \\ b'_1 \end{bmatrix}$$

B

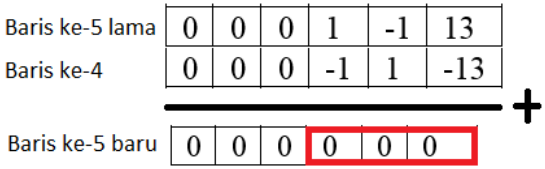
Gambar 5.18 Ilustrasi Matriks Z setelah Iterasi 8 Eliminasi Gauss Sub-Kasus Uji Kedua

Pada iterasi 8 eliminasi gauss, bilangan pada baris ke-4 matriks Z dilakukan operasi baris sehingga nilai bilangan pada baris-4 kolom-3 bernilai 0. Karena bilangan pada baris-4 kolom-3 sudah bernilai 0, maka bilangan pada baris ke-4 sudah tidak perlu diubah lagi. Isi matriks Z setelah iterasi 8 ditunjukkan pada Gambar 5.18.

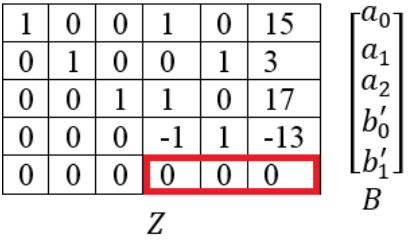
Pada iterasi 9 eliminasi gauss, bilangan pada baris ke-5 matriks Z dilakukan operasi baris sehingga nilai bilangan pada baris-5 kolom-3 bernilai 0. Karena bilangan pada baris-5 kolom-3 sudah bernilai 0, maka bilangan pada baris ke-5 sudah tidak perlu diubah lagi. Isi matriks Z setelah iterasi 9 ditunjukkan pada Gambar 5.19.



Gambar 5.19 Ilustrasi Matriks Z setelah Iterasi 9 Eliminasi Gauss Sub-Kasus Uji Kedua

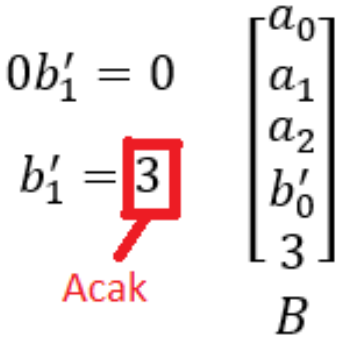


Gambar 5.20 Ilustrasi Penjumlahan tiap Elemen Baris ke-5 Iterasi 10



Gambar 5.21 Ilustrasi Matriks Z setelah Iterasi 10 Eliminasi Gauss Sub-Kasus Uji Kedua

Pada iterasi 10 eliminasi gauss, bilangan pada baris ke-5 matriks *Z* dilakukan operasi baris sehingga nilai bilangan pada baris-5 kolom-4 bernilai 0. Dilakukan operasi penjumlahan pada setiap elemen baris ke-5 dengan setiap elemen baris ke-4 dengan kolom yang sama. Ilustrasi penjumlahan tiap elemen baris ke-5 tertera pada Gambar 5.20. Isi matriks *Z* setelah iterasi 10 ditunjukkan pada Gambar 5.21.



Gambar 5.22 Ilustrasi Matriks B setelah Iterasi 1 Back Substitution Sub-Kasus Uji Kedua

Selanjutnya, dilakukan *back substitution* untuk mencari nilai a_0, a_1, a_2, b_0', dan b_1'. Pada iterasi 1 *back substitution*, semua bilangan pada baris ke-5 bernilai 0. Hal ini dikarenakan sifat dari sistem persamaan linear pada sub-kasus uji kedua yang *linearly dependence*. *Linearly dependence* merupakan kondisi dimana suatu sistem persamaan linear memiliki lebih dari satu solusi. Pada kasus ini, jika variabel b_1' dimasukkan bilangan acak, maka nilai variabel lainnya akan ikut menyesuaikan persamaan linear yang membutuhkan nilai variabel b_1' dan mendapatkan sebuah solusi sistem persamaan linear. Untuk mengisi variabel b_1', maka penulis mengambil angka 3 sebagai bilangan acak. Ilustrasi pengisian nilai variabel b_1' dan terisinya matriks B ditunjukkan pada Gambar 5.22.

Pada iterasi 2 *back substitution*, didapatkan nilai b_0' dengan memasukkan nilai variabel b_1' yang telah dicari pada iterasi sebelumnya ke dalam persamaan $b_0' = \frac{-13-1b_1'}{-1}$ dimana didapatkan hasil $b_0' = \frac{-13-1(3)}{-1} = 16$. Ilustrasi pencarian nilai b_0' dan terisinya matriks B ditunjukkan pada Gambar 5.23.

$$-1b'_0 + 1b'_1 = 0 \qquad \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ 16 \\ 3 \end{bmatrix}$$
$$b'_0 = 16 \qquad B$$

Gambar 5.23 Ilustrasi Matriks B setelah Iterasi 2 Back Substitution Sub-Kasus Uji Kedua

$$1a_2 + 1b'_0 + 0b'_1 = 17 \qquad \begin{bmatrix} a_0 \\ a_1 \\ 1 \\ 16 \\ 3 \end{bmatrix}$$
$$a_2 = 1 \qquad B$$

Gambar 5.24 Ilustrasi Matriks B setelah Iterasi 3 Back Substitution Sub-Kasus Uji Kedua

Pada iterasi 3 *back substitution*, didapatkan nilai a_2 dengan memasukkan nilai variabel b'_0 dan b'_1 yang telah dicari pada iterasi sebelumnya ke dalam persamaan $a_2 = \frac{17-0b'_1-1b'_0}{1}$ dimana didapatkan hasil $a_2 = \frac{17-0-16}{1} = 1$. Ilustrasi pencarian nilai a_2 dan terisinya matriks *B* ditunjukkan pada Gambar 5.24.

Pada iterasi 4 *back substitution*, didapatkan nilai a_1 dengan memasukkan nilai variabel a_2, b'_0, dan b'_1 yang telah dicari pada iterasi sebelumnya ke dalam persamaan $a_1 = \frac{3-1b'_1-0b'_0-0a_2}{1}$ dimana didapatkan hasil $a_1 = \frac{3-3-0-0}{1} = 0$. Ilustrasi pencarian nilai a_1 dan terisinya matriks *B* ditunjukkan pada Gambar 5.25.

$$1a_1 + 0a_2 + 0b'_0 + 1b'_1 = 3 \qquad \begin{bmatrix} a_0 \\ 0 \\ 1 \\ 16 \\ 3 \end{bmatrix}$$
$$a_1 = 0 \qquad B$$

Gambar 5.25 Ilustrasi Matriks B setelah Iterasi 4 Back Substitution Sub-Kasus Uji Kedua

$$1a_0 + 0a_1 + 0a_2 + 1b'_0 + 0b'_1 = 15 \qquad \begin{bmatrix} -1 \\ 0 \\ 1 \\ 16 \\ 3 \end{bmatrix}$$
$$a_0 = -1 \qquad B$$

Gambar 5.26 Ilustrasi Matriks B setelah Iterasi 5 Back Substitution Sub-Kasus Uji Kedua

Pada iterasi 5 *back substitution*, didapatkan nilai a_0 dengan memasukkan nilai variabel a_1, a_2, b'_0, dan b'_1 yang telah dicari pada iterasi sebelumnya ke dalam persamaan $a_0 = \frac{15-0b'_1-1b'_0-0a_2-0a_1}{1}$ dimana didapatkan hasil $a_0 = \frac{15-0-16-0-0}{1} = -1$. Ilustrasi pencarian nilai a_0 dan terisinya matriks B ditunjukkan pada Gambar 5.26.

Setelah semua nilai variabel pada matriks B didapat, maka nilai $f(x)$ yang ditanyakan, yaitu $f(10)$, $f(100)$, $f(1000)$, $f(10000)$, $f(100000)$, $f(1000000)$, $f(10000000)$, $f(100000000)$, dan $f(1000000000)$ bisa dicari nilainya. Proses pencarian nilai $f(10)$, $f(100)$, $f(1000)$, $f(10000)$, $f(100000)$, $f(1000000)$, $f(10000000)$, $f(100000000)$, dan $f(1000000000)$ tertera pada Gambar 5.27.

$f(10) = 0a_0 + 1a_1 + 0a_2 + 1b'_0 + 0b'_1$ $= 0(-1) + 1(0) + 0(1) + 1(16) + 0(3)$ $= 16$
$f(100) = 0a_0 + 1a_1 + 0a_2 + 1b'_0 + 0b'_1$ $= 0(-1) + 1(0) + 0(1) + 1(16) + 0(3)$ $= 16$
$f(1000) = 0a_0 + 1a_1 + 0a_2 + 1b'_0 + 0b'_1$ $= 0(-1) + 1(0) + 0(1) + 1(16) + 0(3)$ $= 16$
$f(10000) = 0a_0 + 1a_1 + 0a_2 + 1b'_0 + 0b'_1$ $= 0(-1) + 1(0) + 0(1) + 1(16) + 0(3)$ $= 16$
$f(100000) = 0a_0 + 1a_1 + 0a_2 + 1b'_0 + 0b'_1$ $= 0(-1) + 1(0) + 0(1) + 1(16) + 0(3)$ $= 16$
$f(1000000) = 0a_0 + 1a_1 + 0a_2 + 1b'_0 + 0b'_1$ $= 0(-1) + 1(0) + 0(1) + 1(16) + 0(3)$ $= 16$
$f(10000000) = 0a_0 + 1a_1 + 0a_2 + 1b'_0 + 0b'_1$ $= 0(-1) + 1(0) + 0(1) + 1(16) + 0(3)$ $= 16$
$f(100000000) = 0a_0 + 1a_1 + 0a_2 + 1b'_0 + 0b'_1$ $= 0(-1) + 1(0) + 0(1) + 1(16) + 0(3)$ $= 16$
$f(1000000000) = 0a_0 + 1a_1 + 0a_2 + 1b'_0 + 0b'_1$ $= 0(-1) + 1(0) + 0(1) + 1(16) + 0(3)$ $= 16$

Gambar 5.27 Ilustrasi Pencarian Jawaban Sub-Kasus Uji Coba Kedua

Jawaban yang dihasilkan sesuai dengan keluaran kasus uji coba pada sumber asli soal yang tertera pada Gambar 5.7 baris kedua untuk sub-kasus uji kedua.

Kemudian perangkat lunak penyelesaian yang diimplementasi pada BAB 4 dijalankan dengan masukan pada contoh kasus uji coba pada Gambar 5.1 dan hasil keluaran sesuai seperti hasil keluaran contoh kasus uji coba pada Gambar 5.7. Hasil berjalannya perangkat lunak yang dijalankan tertera pada Gambar 5.28.

```
2
2
5 7 5 7
6 7 8 9
5 7 5 7
3
15 3 17 2 16 4 15 3 17
10 100 1000 10000 100000 1000000 10000000 100000000 1000000000
16 16 16 16 16 16 16 16 16

--------------------------------
Process exited after 3.688 seconds with return value 0
Press any key to continue . . .
```

Gambar 5.28 Masukan dan Keluaran pada Perangkat Lunak Berdasarkan Contoh Kasus Uji Kebenaran

Selanjutnya akan dilakukan uji coba sebanyak 20 kali dengan membandingkan hasil keluaran perangkat lunak dengan hasil sebenarnya. Terlebih dahulu dijalankan sebanyak 20 kali data *generator* skenario 1. Setelah data *generator* skenario 1 dijalankan, maka akan muncul 20 berkas masukan acak dan 20 berkas keluaran benar sebagai pembanding. Setelah itu, perangkat lunak hasil implementasi dijalankan sebanyak 20 kali dengan menggunakan 20 berkas masukan sebagai data masukan. Kemudian akan muncul 20 berkas keluaran perangkat lunak hasil implementasi yang akan dibandingkan dengan 20 berkas keluaran benar. Tabel 5.1 menunjukkan bahwa solusi yang diimplementasikan pada perangkat lunak adalah benar karena tidak ada perbedaan angka pada berkas keluaran hasil solusi dengan berkas keluaran benar dan menghasilkan akurasi 100%.

5.2.2 Uji Coba Kinerja

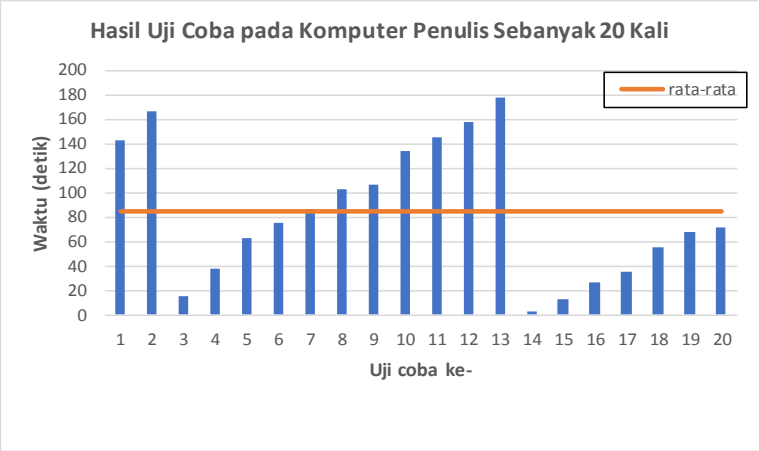
Selanjutnya akan dilakukan uji coba kinerja atau waktu berjalan perangkat lunak. Kompleksitas total berjalannya perangkat lunak adalah $O(TN^6 + TN^3)$ dimana T adalah banyaknya sub-kasus uji coba dan N adalah banyaknya urutan berkala awal. Karena terdapat masukan banyak sub-kasus uji, maka perlu adanya dua skenario pengujian. Uji coba skenario pertama dilakukan untuk mengetahui rata-rata kinerja atau waktu berjalan perangkat lunak keseluruhan,

sedangkan uji coba skenario kedua dilakukan untuk mengetahui pengaruh banyak urutan berkala awal N terhadap pertumbuhan waktu berjalan perangkat lunak.

Tabel 5.1 Percobaan Perbandingan Jawaban Solusi dengan Jawaban Benar

Uji coba ke-	Total selisih angka keluaran hasil solusi dan keluaran benar	Jumlah jawaban $f(x)$ sesuai per jumlah jawaban total	Akurasi
1	0	61000/61000	100%
2	0	71500/71500	100%
3	0	6422/6422	100%
4	0	17742/17742	100%
5	0	26487/26487	100%
6	0	32334/32334	100%
7	0	37199/37199	100%
8	0	44149/44149	100%
9	0	49108/49108	100%
10	0	56397/56397	100%
11	0	61681/61681	100%
12	0	64381/64381	100%
13	0	71981/71981	100%
14	0	1597/1597	100%
15	0	5745/5745	100%
16	0	11031/11031	100%
17	0	16306/16306	100%
18	0	22425/22425	100%
19	0	28217/28217	100%
20	0	31381/31381	100%

Uji coba skenario pertama dilakukan dengan menjalankan perangkat lunak hasil implementasi sebanyak 20 kali dengan menggunakan 20 data masukan yang telah dibuat pada uji coba



Gambar 5.29 Hasil Uji Coba Kinerja Perangkat Lunak Sebanyak 20 Kali

kebenaran. Uji coba dilakukan pada komputer penulis. Uji coba akan dilakukan sebanyak 20 kali dengan menggunakan 20 data masukan sebelumnya dan menghasilkan 20 hasil uji coba kinerja atau waktu berjalan. Hasil uji coba yang tertera pada Gambar 5.29 dan Tabel 5.2 menunjukkan bahwa waktu eksekusi minimal perangkat lunak adalah 3,482 detik, waktu eksekusi maksimal adalah 178 detik, dan waktu eksekusi rata-rata adalah 84,2321 detik.

Uji coba skenario kedua dilakukan dengan menjalankan data *generator* skenario 2 yang tertera pada Kode Sumber 4.18 untuk membuat data masukan acak sesuai dengan batasan variabel pada soal. Pada uji coba skenario kedua, banyak sub-kasus uji yang digunakan hanya satu buah saja agar dapat melihat pengaruh banyak urutan berkala awal N terhadap pertumbuhan waktu. Nilai N yang diberikan bukan berasal dari angka acak, tetapi nilai N dimasukkan sendiri oleh pengguna. Dalam hal ini, penulis berlaku sebagai pengguna. Data *generator* skenario 2 dijalankan sebanyak

14 kali masing-masing dengan nilai N yang berbeda sesuai dengan banyak nilai N yang mungkin menjadi masukan perangkat lunak. Nilai N yang mungkin menjadi masukan dimulai dari $N = 1$, $N = 2$, $N = 3$, hingga $N = 14$.

Tabel 5.2 Hasil Uji Coba Kinerja Perangkat Lunak Sebanyak 20 Kali

Uji coba ke-	**Waktu (detik)**
1	143,3
2	166,1
3	15,1
4	37,65
5	62,96
6	75,4
7	86,27
8	102,8
9	106
10	133,8
11	145,4
12	158,3
13	178
14	3,482
15	12,92
16	26,72
17	35,77
18	55,58
19	67,66
20	71,43

Hasil uji coba skenario kedua yang tertera pada Gambar 5.30 dan Tabel 5.3 menunjukkan bahwa semakin banyak urutan berkala N, semakin besar pertumbuhan waktu perangkat lunak.

Gambar 5.30 Hasil Uji Coba Pengaruh Banyak Urutan Berkala Awal Terhadap Pertumbuhan Waktu

Tabel 5.3 Hasil Uji Coba Kinerja Skenario 2

Banyak Urutan Berkala Awal *N*	**Waktu (detik)**
1	0
2	0
3	0
4	0,01
5	0,01
6	0,01
7	0,03
8	0,05
9	0,09
10	0,16
11	0,27
12	0,33
13	0,63
14	0,83

BAB 6
KESIMPULAN DAN SARAN

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan dan saran mengenai hal-hal yang masih bisa untuk dikembangkan dari tugas akhir ini.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap implementasi penyelesaian permasalahan prediksi hasil penjumlahan N urutan berkala, dapat diambil kesimpulan sebagai berikut:

1. Implementasi metode eliminasi gauss menghasilkan jawaban prediksi yang tepat untuk 20 kasus acak.
2. Waktu rata-rata yang dibutuhkan oleh perangkat lunak untuk menyelesaikan permasalahan masih sangat lama dan diatas 1 detik.
3. Waktu yang dibutuhkan oleh perangkat lunak untuk melakukan penyelesaian permasalahan mengalami pertumbuhan secara eksponensial pangkat 6 terhadap pertumbuhan banyak urutan berkala awal N.
4. Kompleksitas waktu yang dibutuhkan untuk seluruh proses perangkat lunak adalah $O(TN^6 + TN^3)$ pada banyak sub-kasus uji T dan banyak urutan berkala awal N.

6.2 Saran

Saran untuk pengembangan selanjutnya yaitu perlu adanya optimasi waktu dengan mengubah permasalahan ke dalam bentuk interpolasi trigonometri yang diselesaikan dengan metode interpolasi polinomial Lagrange dan perkalian polinomial yang diselesaikan dengan metode transformasi Fourier cepat.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] SPOJ, "PERIOD4 - Periodic function, trip 3 (easy)," 2015. [Online]. Available: http://www.spoj.com/problems/PERIOD4/. [Diakses 1 Juni 2017].

[2] NOAA, "Solar Cycle Progression," [Online]. Available: http://www.swpc.noaa.gov/products/solar-cycle-progression. [Diakses 13 Juli 2017].

[3] A. J. Menezes, P. C. v. Oorschot dan S. A. Vanstone, Handbook of Applied Cryptography, Boca Raton: CRC Press, 1997.

[4] H. Anton dan C. Rorres, Elementary Linear Algebra Ninth Edition Applications Version, Hokoben: John Wiley & Sons, Incorporated, 2003.

[5] S. C. Chapra dan R. P. Canale, Numerical Methods for Engineers Sixth Edition, New York: McGraw-Hill Education, 2009.

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Daniel Henry, dilahirkan di Jakarta pada tanggal 17 September 1994. Penulis menempuh jenjang pendidikan sekolah dasar di SD Santa Lusia Bekasi pada tahun 2000 hingga tahun 2006, sekolah menengah pertama di SMP Santa Lusia Bekasi pada tahun 2006 hingga tahun 2009, dan sekolah menengah atas di SMA Marsudirini Bekasi, Jawa Barat, pada tahun 2009 hingga tahun 2012. Setelah menempuh jenjang SMA, penulis menempuh jenjang S1 jurusan Teknik Informatika di Institut Teknologi Sepuluh Nopember (ITS) Surabaya sejak tahun 2012 hingga sekarang. Selama menjalani masa perkuliahan, penulis telah terlibat dalam beberapa organisasi mahasiswa. Pertama, penulis pernah menjabat sebagai staf departemen riset dan teknologi Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) ITS pada tahun 2013 hingga tahun 2014. Kedua, penulis pernah menjabat sebagai staf departemen riset dan teknologi Badan Eksekutif Mahasiswa Fakultas Teknologi Informasi (BEM FTIf) ITS pada tahun 2014 hingga tahun 2015.